DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Modern Global Optimisation Heuristics in the Long Term Planning of Telecommunication Networks

A thesis submitted as a requirement for the degree of Master of Engineering in Electronic Engineering

June 1995

Supervisor: Dr. T. Curran

James McGibney
B.E.

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master of Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: James McGibney **ID No.:** 92701060

Date: 4/9/95

Acknowledgements

Very special thanks are due to Dr Tommy Curran for giving me the opportunity to pursue this research, and for his expert guidance, flexibility and good humour throughout A word of thanks is also due to Tommy for providing opportunities to travel to present my work and make external contacts

I also wish to sincerely thank Dr Dmitri Botvich for his endless patience in attempting to deepen my knowledge and appreciation of mathematical methods, and for his constant availability and enthusiasm for offering advice I would also like to thank John Murphy and Ciarán Treanor for their help, especially in the early stages

Finally, I wish to thank my parents for their constant interest and encouragement and the friends I have made here at DCU for making my time here very enjoyable and rewarding

Title: **Modern Global Optimisation Heuristics in the Long Term Planning of Telecommunication Networks**

Author: **James McGibney**

Abstract:

Telecommunication transmission networks are faced with greatly increasing bandwidth demands due largely to a growth in the number of subscriber services on offer and the nature of these services Existing (and new) operators are presented with the problem of installing new capacity to cater for this The scale of these networks and the immense investment being undertaken suggests that much attention in planning be given to minimum cost design

This thesis is concerned with the area of network topological optimisation Network topological design problems require the optimal configuration of a network to meet a set of requirements while minimising total cost.

Network topological optimisation problems are usually computationally difficult Many such problems belong to the *NP*-hard class of problems, for which no polynomial-time solution algorithms have been found As there are frequently a large number of diverse local optima, the use of pure local optimisation strategies can be rather inefficient to search for the global optimum or a feasible solution close to it

The major concern of this work is to consider some approaches to global optimisation based on a non-trivial combination and adaptation of modern optimisation heuristics, namely *tabu search*, *simulated annealing* and *genetic algorithms* The performance of these algorithms is evaluated with reference to quality of solution obtained, robustness, and time taken to converge. Existing methods are also reviewed and improvements in performance are achieved

To ensure that our tests are independent of the idiosyncrasies of any particular network, a mechanism for generating random, realistic test problems is developed A classification of problems is introduced to examine how algorithm performance depends on the nature of the problem We see that a parameter defining this classification is in fact closely related to the number of local optima in the problem

Table of Contents

1. Introduction **4**

1 1 Introduction 4

1 2 Objective 4

1 3 Thesis Structure 5

1 4 Associated publication 6

2. Network Topological Design **7**

2 1 Introduction 7

2 2 Problem Formulation Review of Modelisations in the Literature 9

2 2 1 Input Data 9

2 2 2 Representation of Solutions 10

2 2 3 Constraints 10

2 2 4 Cost Function Models 12

2 3 Review of solution approaches in the literature 14

2 3.1 Solution set size and structure 14

2 3 2 Previous Solution approaches 15

2 3 3 New techmques 17

2 4 Combinatorial Optimisation 17

3. Complexity Issues: Reformulation as a Combinatorial Optimisation Problem **19**

3 1 Decomposition of the Problem 20

3 2 The Question of Splitting 20

3 2 1 Discrete link cost function 20

3 2 2 Concave link cost function 21

3 2 3 Concave with fixed cost case 23

3 2 4 Linear with fixed cost case 23

3 3 Finding an Optimal Routing 23

3 4 Finding an Optimal Link Arrangement, A Combinatorial Optimisation Problem 25

3 4 1 Complexity 25

3 5 Conclusion 25

4. Solution Methods **27**

4 1 Greedy Algorithms 27
4 1 1 Standard Greedy Algorithm 27
4 1 2 Accelerated Greedy Algorithm 29
4 1 3 An Example 31
4 1 4 'Pure' Greedy algorithm 33
4 2 Local Search Methodology 34
4 3 Modern Approaches 36
4 3.1 Simulated Annealing 36
4 3 2 Tabu Search 38
4 3 3 Genetic Algorithms 40
4 4 Tabular Comparison of Methods 41

5. Implementation **43**

5 1 General Issues 43
5 1 1 Data Structures and Memory Considerations 43
5 1 2 Random Number Generator 46
5.2 Shortest Path Algorithms 46
5 2 1 Single pair shortest path algorithms 46
5 2 2 All shortest paths 46
5 2 3 Improvements in efficiency - literature 47
5 2 4 Improvements in efficiency - Updating all shortest paths 47
5 3 Generation of Test Problems 50
5.3 1 Classification of Fixed Charge Problems 50
5 3 2 Generation of Test Problems 51
5 3 3 Parameter choices 52
5 4 Implementation of Solution Strategies 53
5 5 Greedy Algorithms 53
5 6 Tabu Search 54
5 6 1 Short-term memory 54
5.6 2 Longer-term memory 55
5 7 Simulated Annealing 57
5 7.1 Cooling Schedule 58
5 8 Genetic Algorithms 59
5 8 1 The Simple Genetic Algorithm 59
5 8 2 An adaptation of genetic algorithms to our problem 60
5 9 Conclusion 62

6. Results and Comparisons **63**

6 1 Introduction 63

6 2 Comparison of Greedy Algorithms 64
6 2 1 Comparison of results for greedy algorithms 64
6 2 2 Performance improvement with the Accelerated Greedy Algorithm 66
6 2.3 Conclusion 66
6 3 Some Tabu Search Strategies 67
6 4 Cooling Schedules for Simulated Annealing 70
6 5 Genetic Algorithms and an Adaptation 73
6 5 1 Pure Genetic Algorithm 73
6 5.2 Hybrid Greedy-Genetic Algorithm 73
6 6 Overall Comparison of Local Search Methods 76

7. Conclusion 79

7 1 Overall Discussion of Results 79
7 1 1 Comparison of Local Search Methods 79
7 1 2 Effects of k_{char} on performance 80
7 1 3 Conclusions from experimental work 80
7 2 Applications and Scope for Further Work 81
7 2 1 Direct application to network planning 81
7 2 2 Application of algorithms in other areas 81
7 2 3 User interface development 81
7 2 4 Further refinement of the algorithms 82
7 2 5 Other methods 82
7 3 Summary of Overall Conclusions 82

References 84

Appendix 1: Shortest Path Algorithms A-1

A1 1 Shortest path between two specific nodes Dijkstra's algorithm A-1
A1 2 Shortest paths between all node pairs Floyd-Warshall algorithm A-2

Chapter 1. Introduction

1.1 INTRODUCTION

Telecommunication transmission networks are faced with greatly increasing bandwidth demands due largely to a growth in the number of subscriber services on offer and the nature of these services. Existing (and new) operators are presented with the problem of installing new capacity to cater for this Most of this growth is in the area of data communications, and is especially due to an increase in the demand for multimedia services among both business and domestic users Future growth in bandwidth demand seems assured for the following major reasons

- There is substantial growth in low-cost computing capability among the general public as personal computers grow from relatively small machines to machines with enormous computing power
- Increasing deregulation and competition among network operators is encouraging diversification into the provision of new high-bandwidth consumer multimedia services such as home shopping and video-on-demand
- There is an ever-increasing need for data interchange in the business world and there is substantial growth in demand for high bandwidth services like video conferencing

The scale of these networks and the immense investment being undertaken suggests that much attention in planning be given to minimum cost design This thesis is concerned with the area of network topological optimisation Network topological design problems require the optimal configuration of a network to meet a set of requirements while minimising total cost Topological design methods can be applied equally to the design of new physical networks and to the design of increasingly popular private leased line networks

1.2 OBJECTIVE

Network topological optimisation problems are usually computationally difficult Many such problems belong to the *NP*-hard class of problems that cannot be solved in realistic

time on modern computers As there are frequently a large number of diverse local optima, the use of pure local optimisation strategies can be rather inefficient to search for the global optimum or a feasible solution close to it Existing network design algorithms use local optimisation heuristics The major objective of this thesis is to review these existing algorithms with a view to achieving improvements in performance and to investigate the use of newly developed global optimisation algorithms The particular algorithms under investigation are simulated annealing, tabu search and genetic algorithms These algorithms have already been used with considerable success to solve a wide range of complex problems in a variety of different areas

1.3 THESIS STRUCTURE

An introduction to the terminology and ideas of network topological optimisation is given in Chapter 2 This is followed by a formal description of the general problem that is consistent with a variety of problems discussed in the literature. A review of existing literature on the solution of these problems is also presented Finally, the general area of combinatorial optimisation is introduced

Chapter 3 is concerned with a complexity study and a particular difficult but reasonably well-behaved approximate model is selected for further investigation In this complexity study the problem is decomposed and subproblems identified

A variety of solution methods are introduced and discussed in Chapter 4 These include greedy algorithms as well as modern global optimisation procedures such as the methods of simulated annealing, genetic algorithms and tabu search

Chapter 5 is concerned with the implementation of the algorithms Details of the adaptation and use of all the solution methods discussed in the previous chapter are presented The generation of realistic test problems and the use of shortest path algorithms are also discussed A particularly efficient use of shortest-path algorithms is presented in Section 5 2 A new hybrid combination of a greedy algorithm and a genetic algorithm is dicussed in Section 5.8 2

Results are presented and discussed in Chapter 6 Results are firstly presented for parametric studies of each of the methods under consideration Finally, an overall comparison of methods is presented.

An overall discussion of results, applications, and scope for further work is given in Chapter 7 Finally, overall conclusions are summarised A successful application in a decision support system is briefly described

Appendix 1 provides formal descriptions of the shortest-path algorithms that are used

1.4 ASSOCIATED PUBLICATION

A summary of part of this work was presented at ATNAC'94, the Australian Telecommunication Networks and Applications Conference in November 1994 [27]

Chapter 2. Network Topological Design

This chapter is primarily concerned with the introduction of network topological design and optimisation to the reader and a review of existing literature on the subject Terminology and ideas are introduced in Section 2 1 Section 2 2 discusses a formal model which is sufficiently general to be consistent with a variety of problems discussed in the literature Solution methods used for these various problems are described in Section 2 3 and the existence of a new class of (quite different) methods, with proven success in other areas, is discussed Section 2 4 briefly introduces combinatorial optimisation to facilitate better understanding of Chapter 3

2.1 INTRODUCTION

Network topological design problems require the optimal configuration of a network to meet a given set of requirements while minimising total cost As well as in telecommunication networks, with which we are concerned, such problems occur in areas as diverse as transportation, energy and other distribution networks

In the context of telecommunications we are generally given a set of *N nodes* (Figure 2 1) To the designer of a national trunk network, the nodes would represent cities or principal exchanges, to a private network planner, they would represent the offices that are to be connected These nodes have to communicate with one another, exchanging telephone traffic (i e voice and data) The purpose of a network is to facilitate communication by means of *links* installed between some chosen pairs of nodes A solution network could (in theory) be a minimum network (Figure 2 2) with large capacity assignments on each link The other extreme is a fully connected network (Figure 2 3) with smaller capacity assignments An optimal network configuration will most likely be of the type shown in Figure 2 4, due to a trade-off between various cost and constraint elements

Specification of where links are installed is of course insufficient to describe a telecommunications network A parameter representing the amount of resources to be

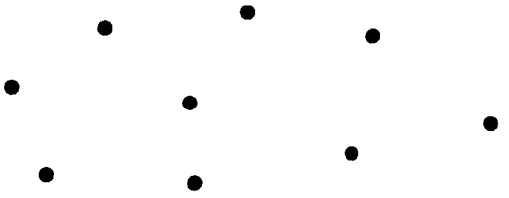


Figure 2.1 Given set of nodes

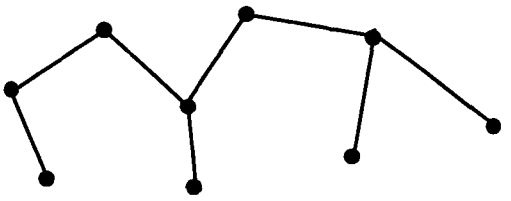


Figure 2.2 A minimum network

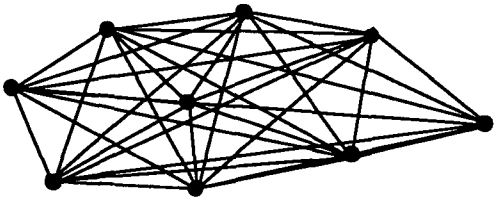


Figure 2.3 Fully connected network

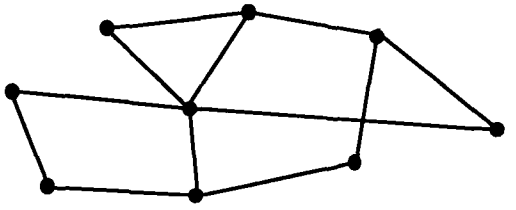


Figure 2.4 Something in between

made available on each link of the network must be specified This will be the link *capacity*, or bandwidth

As well as node locations, parameters giving some measure of the intensity of communication to be established between pairs of nodes must be specified for optimal network design For each pair of nodes, these communication needs are frequently modelled as flow requirements, or *demands* The demand between a pair of nodes can be defined as the capacity to be installed between the node pair, either on a single path (possibly just a single link) or on several separate paths Demands between node pairs will of course have the same units as capacity (e g Mb/s)

Demands are determined with reference to predicted traffic between pairs of nodes, and are assigned values so that a certain quality of service is maintained This quality of service is realised by ensuring that blocking probability is kept to an acceptably low level Blocking may occur if there is insufficient available capacity to satisfy a demand at a point in time Traffic predictions can be made on the basis of existing patterns but social and technological factors must be accounted for in long term planning A detailed discussion on practical demand forecasting is given in [26]

The network design will generally be subject to certain conditions, particularly reliability, performance and capacity constraints The requirement for conservation of flows at nodes can also be considered as a constraint on the design This requirement is that capacity into a node should equal capacity out of the node

The following section gives a more mathematical description of this type of problem and discusses some cost function models

2.2 PROBLEM FORMULATION: REVIEW OF MODELISATIONS IN THE LITERATURE

In any design problem it is important to know what part of the overall design process concerns us, i.e what information will be given and what is the required output? In our optimisation problem formulation, we shall assume that we are given certain data Our output will be the optimal values of specified variables, or at least better values than have been achieved before

An initial broad description is useful In an optimisation problem, we assume that we are given

(1) Input data
(2) The solution definition - i e what is required?
(3) Conditions that a set of output variable values must meet for the solution they represent to be considered valid
(4) A method of evaluating the cost of a given solution, i e an objective function We wish to minimise this cost

In the following sections, a network topological optimisation model is presented in accordance with the Cross Connect Networks (CCN) model of [4] Most of the literature in optimal network topology design is consistent with this model, with individual researchers looking at specific problems Reviews by Minoux [30], Boorstyn & Frank [3] and Gerla & Kleinrock [9] discuss a range of such specific problems In our formulation, reference is frequently made to these papers and other work in the area

2.2.1 Input Data

Let $V = \{1, \quad, N\}$ be a (given) set of nodes and let L be the set of all pairs of nodes $l = (i, j)$ Let $L_b \subset L$ be the set of node pairs representing the *possible* links which can be included in a network configuration Denote the graph $G_b = (V, L_b)$ the *base graph*

The locations of these nodes will have already been determined and we can assume that we are given a matrix of inter-node *distances*, $D = (d_{i,j})_{i,j=1, \quad, N}$ An element of the distance matrix corresponds to the length of a link between i and j, should one be installed We can expect the cost of installing such a link to be related to this distance

We will also have a matrix of inter-node *demands*, $R=(r_{i,j})_{i,j=1,\ldots,N}$ The demand, $r_{i,j}$, refers to the capacity, or bandwidth, to be installed between nodes i and j, either directly or via other nodes

We assume the following, $\forall\, i,j \in V$

$$\begin{aligned} d_{i,j} &> 0, \text{ if } i \neq j \qquad & r_{i,j} &\geq 0, \text{ if } i \neq j \\ &= 0, \text{ if } i = j & &= 0, \text{ if } i = j \end{aligned} \tag{2 1}$$

We do not assume that matrices D and R are symmetric, although matrix D would normally be so Neither do we assume that matrix D satisfies *the triangle inequality*[1].

Some work has been done in topological design where optimisation is performed with respect to performance measures [9] In [10] Gersht & Weihmayer place upper bounds on link utilisation We can assume in our case, however, that such issues are already dealt with in the determination of demand values

2.2.2 Representation of Solutions

Our objective is to find a minimum cost network implementation We define a *solution* as a topology, i e any link arrangement with capacity assignments In other words, between which node pairs do we install links, and what capacities should these links have? We define an *adjacency* matrix, $A=(a_{i,j})_{i,j=1,\ldots,N}$, and a *capacity* matrix, $C=(c_{i,j})_{i,j=1,\ldots,N}$ to represent a solution The adjacency matrix describes the link arrangement $a_{i,j}$ equals 1 if link (i,j) is present and equals 0 otherwise

2.2.3 Constraints

It is necessary that the demands are satisfied by the allocation of capacities in the network This requirement can be stated mathematically in terms of a multicommodity flow model [4] [30] Here, we use the closely connected CCN model of [4] In [25], Lee et al justify using this type of model on the basis that it has fewer constraints

For any node pair $(i,j) \in L$, let $P_{i,j}$ be a set of paths from i to j, where any path $p \in P_{i,j}$ is a sequence of nodes, $i = i_0, i_1, \ldots, i_n = j$ To satisfy our demands, we require, $\forall\,(i,j) \in L$,

---

[1] A matrix D satisfies the triangle inequality if, for any $i,j,k \in V$,

$$d_{i,j} + d_{j,k} \geq d_{i,k}$$

$$r_{i,j} = \sum_{p \in P_{i,j}} c_p , \tag{2 2}$$

where $c_p \geq 0$ is the capacity of path p To do this we need, for all links $l = (i, j)$,

$$c_{i,j} \geq \sum_{p \in T_{i,j}} c_p , \tag{2 3}$$

where $T_{i,j}$ is a set of all paths using link (i,j) and $C = \left(c_{i,j}\right)_{i,j \in V}$ is the capacity matrix defined in Section (2.2). If capacity is a continuous variable, taking any (positive) value, then Equation (2 3) can be written as an equality

The above are necessary constraints to ensure that demands are satisfied A variety of other constraints have been considered for problems of this type and we would like our formulation and solution methods to be sufficiently flexible to allow for easy constraint imposition Such constraints could include, but are not limited to, the following

- Reliability constraints, what happens if one or more nodes or links fail? *Connectivity tests* can be imposed on a graph to ensure survival on the failure of any specified number of nodes and links
- Performance constraints, as mentioned in Section 2 1, performance issues related to traffic intensity are not of concern as they will have been dealt with at the demand forecasting stage A constraint on the total number of links in a path could be imposed to reduce switching and propagation delays.
- Capacity constraints (upper bounds), Some researchers have placed upper bounds on link capacities in their models (e g [9] [10], Gersht & Weihmayer point out that these upper values can be set at large values for unconstrained allocation) It is felt that such constraints are not necessary in our model, there should be no constraint on conduit sizes in the transit network and an arbitrarily large number of cables could be installed

It is best for us, however, to keep our formulation as general (and straightforward) as possible at first In [10] it is suggested that optimisation be firstly performed with simple constraints with minimum cost corrections being made to the final topology to satisfy other conditions We will however add a constraint that the adjacency matrix, A, is symmetric, we expect that the initial costs incurred in setting up a two-way link are not much greater than those incurred with a one-way link (i e cost of digging, getting right-of-way, etc)

2.2.4 Cost Function Models

We need to be able to express the cost of a given solution, $s \in S$, in terms of matrices A and C (Section 2.2):

$$\text{TOTALCOST}(s) = F(A,C). \tag{2.4}$$

The nature of this cost function will determine how difficult the optimisation problem is.

We can express the cost of installing a network in terms of elemental costs. In a paper by Gersht & Weihmayer [10], link and node costs are included:

$$F(A,C) = \sum_{\substack{\text{all nodes} \\ n \in V}} (\text{node costs}) + \sum_{\substack{\text{all links} \\ l=(i,j) \in L_b \\ \text{with } a_{i,j}=1}} (\text{link costs}) \tag{2.5}$$

Both node and link costs will be functions of capacity switched or routed, so we re-write 2.5 as:

$$F(A,C) = \sum_{\substack{\text{all nodes} \\ n \in V}} f_n(c_n) + \sum_{\substack{\text{all links} \\ l=(i,j) \in L_b}} f_l(c_l).a_l \tag{2.6}$$

It is often possible, however, to model all or part of the cost of nodes as link costs [21]. This is because the capacity switched at a node is related to that on the links emanating from the node. Initial node costs do not concern us as all N nodes will be present in all solutions under consideration.

For these reasons we assume that the cost of a given solution can be expressed as a sum of link costs and we have:

$$F(A,C) = \sum_{\substack{\text{all } l=(i,j) \\ i,j=1,\cdots,N}} f_l(c_l).a_l, \tag{2.7}$$

which is consistent with Minoux [30].

A review of the published work suggests that we consider four link cost functions as shown in Figure 2.5.

The link cost function of Figure 2.5(a), where cost is a discontinuous function of capacity, most closely represents reality. This is because cables are usually available with discrete capacities only. The other three cost functions of Figure 2.5 are approximations to this; a link cost function will always be a nondecreasing function of capacity, of course. In [20], Kerner et al discuss real costs associated with the installation of both metallic and optical fibre facilities on an interoffice network. We can expect to encounter similar cost types when dealing with the transmission network. In both cases a large cost is associated with channel construction and much of the remainder consists of cable costs (including placement) which will depend on capacity.

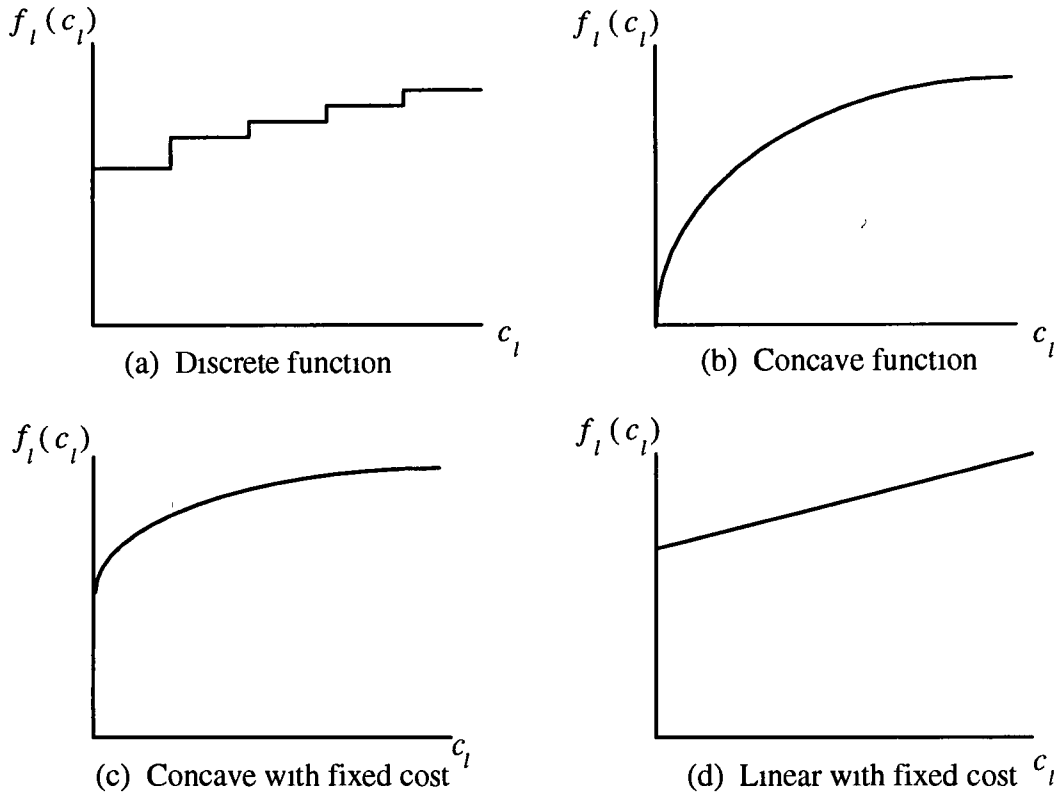


Figure 2.5 *A typical link cost function (a) and some approximations (b), (c), (d)*

They assume that construction cost is proportional to distance, while acknowledging that this is a gross assumption Digging up streets will be more expensive in urban than rural areas and also will depend on the terrain All we need for our model is a total construction cost for a particular link, so we do not need to make that assumption Cable costs will depend on distance, but again, as distance is constant for a particular link, this is not so important Link cost will depend on capacity, however, as capacity determines the number, size and type of cables required

Some researchers have developed heuristic methods for topological optimisation where the link cost is a staircase function of capacity of the type shown in Figure 2 5(a) [8][21] More frequently, however, the approach of Gerla & Kleinrock [9] is taken where, "for computational efficiency, it is often convenient to approximate discrete costs with continuous costs during the initial optimisation phase, and to discretise the continuous values during a refinement phase" This is acceptable if the link flows are sufficiently large, i e if the expected 'error' in discretisation is small A justification for using the contmuous approximation, from the point of view of computational efficiency, is given in Section 3 2

If an approximate continuous cost function is to be considered, its shape should closely represent reality In [35], Yaged suggests using a concave function (Figure 2 5(b)), which he justifies by referring to the *economies of scale*[2] phenomenon This cost function is also considered as the continuous approximation in [28] and [37] and is one of those considered in [9] and [30] The relationship of Figure 2 5(c) is also discussed by Yaged and is more realistic than (b)

Another approximation is shown in Figure 2 5(d) Here each link cost consists of an initial cost of opening the link and a part that is linearly proportional to the capacity to be installed (the *fixed charge problem*), This type of cost function is widely used [9][10][30] and is justified by the large start-up cost associated with link installation This cost model is used in solving network design problems in areas other than telecommunications, particularly in road network design, and a rich literature exists [2][16][23][33]

It is worth noting that the functions shown in Figure 2 5(b) and (d) are special cases of that in (c).

2.3 REVIEW OF SOLUTION APPROACHES IN THE LITERATURE

Before looking directly at solution approaches, the structure of the solution set is worthy of consideration It will be seen in later chapters that the number of local optima and their proximity (in terms of cost) to the global optimum influences the relative effectiveness of our algorithms

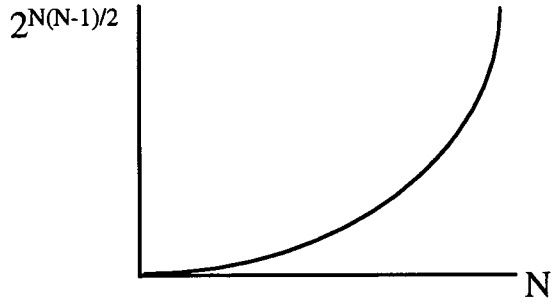
2.3.1 Solution set size and structure

We have defined a solution in terms of matrices A and C We define a *feasible solution* as a pair (A,C) that satisfies all constraints The *solution space, S,* will be the set of all possible feasible solutions. A is a matrix of discrete variables and C is in general a matrix of continuous variables, so the solution space is continuous In many situations, however, the solution space can be viewed as discrete if optimal capacity assignment always results in only one path being used for each node pair

Even with these reduced problems, the solution set is very large A can range from the representation of $N-1$ two-way links to that of the fully connected base graph, G_b, with up to $N(N-1)/2$ two-way links Figures 2 2 and 2 3 show these two extremes

[2]Well-known economic concept of decreasing marginal cost with increasing size

for a nine-node example The number of possible link arrangements is very large for any appreciable N (of order 2^{N^2}) In the following chapter, our discussion on choice of cost function model is influenced by the nature of the solution space implied by each model Figure 2 6 shows how the solution space size increases with N



N	$2^{N(N-1)/2}$
5	~ 10^3
10	~ 10^{13}
15	~ 10^{31}
20	~ 10^{57}
25	~ 10^{90}

Figure 2.6 *Increase in number of solutions with number of nodes*

As most methods in the literature do not claim to find the global optimum, but rather a 'good' local optimum, we would like to know how good we can expect local optima to be In A D Pearman's 1979 paper [33], the structure of the solution set to a number of combinatorial problems is considered, with particular emphasis on the road network optimisation problem (RNOP) which is similar to our current problem The frequency distributions of objective function values are determined for small problems and it is found that the distribution for the RNOP is especially *positively skewed* - i e it possesses a large number of good sub-optimal solutions The frequency distribution, of course, says nothing about solution technique and the existence of many good solutions does not mean that all techniques will be capable of finding one It is likely that this property would also hold for larger networks but it is not clear what would happen if the set of feasible solutions was reduced in size by the imposition of reliability or performance constraints

2.3.2 Previous Solution approaches

The fixed charge problem is among those considered by Minoux in his 1989 survey paper [30]. He firstly refers to a *branch and bound* method which provides an exact solution for small networks ($N < 10$ or 15), but is not practical for larger networks as implicit enumeration of all feasible solutions is required A more interesting *greedy* approach, a local optimum finder that has shown promising results, is presented and applied to concave and fixed charge models This involves selecting a solution (perhaps a network with all possible links in place) and removing links one by one until no further removal causes a decrease in cost The link to be removed at a particular iteration is that which will bring about the greatest cost reduction when removed A

much more efficient *accelerated greedy algorithm*, which works on the same basis, is also given The implementation of these algorithms is discussed further in Chapters 4 and 5 and experimental results are given in Chapter 6

In a 1990 paper [10], Gersht & Weihmayer look at a similar problem, although they are also concerned with other issues in their optimisation A greedy-type algorithm, with link removal only, is also considered here Use of this type of algorithm is justified on the basis of computational experience, consistent with that of Pearman (above), that 'in many network design cases, global minima are relatively flat, and all local minima lead to about the same design cost' They also remark that a greedy algorithm of this type has the advantage of easy adaptability to specific model requirements They suggest some methods of accelerating the algorithm and note the benefit of good use at a particular iteration of information obtained during previous iterations These acceleration approaches are further discussed in the next section

Two papers from the 1970s, one by Boorstyn & Frank [3] and another by Gerla & Kleinrock [9], consider a problem of our type as part of a broader network design study, where other issues, such as node selection and location in the former and questions of delay (in packet networks) in the latter, are dealt with. The branch exchange approach of Boorstyn & Frank is also a greedy-type algorithm Although in the area of transportation, a 1973 paper by Billheimer & Gray [2] is also of interest as it is concerned with our fixed charge problem Again, a greedy approach is taken where *both* the deletion of uneconomic links and the insertion of economic links are considered The algorithm is aided by the insertion of upper and lower bounds on the fixed and variable portions of the global optimum and criteria are determined for nominating links as *strongly* acceptable or unacceptable

Although a different (discrete) cost function is used, the approach of Kershenbaum et al in their 1991 paper [21] is noteworthy The general approach is to see what characteristics a good design could be expected to have (e g requirements will follow fairly direct rather than circuitous paths and paths will have a small number of links) An attempt is made to incorporate these desirable features to the greatest extent possible, and a heuristic method is proposed The idea is that a tree[3] is selected and traffic is sent over a direct route if the requirement is sufficiently large and towards the centre of the tree otherwise This approach is somewhat inexact and is considered good for achieving fast results of indifferent quality, perhaps to be used as starting solutions for another heuristic

[3]A *tree* is a connected graph (or network) without cycles and contains N-1 links

2.3.3 New techniques

It is noteworthy that, with the possible exception one very approximate approach, all established methods considered for a variety of network topological design problems are effectively greedy algorithms

One of the principal aims of this work is to investigate new methods for this type of problem A number of very interesting methods have appeared relatively recently in the area of *combinatorial optimisation* These methods have been successful in the solution of a wide variety of very difficult problems, some quite similar in complexity to typical network design problems A brief introduction to combinatorial optimisation is given in the next section

Three of these general methods are *simulated annealing, tabu search* and *genetic algorithms* Evidence of the success of these methods in areas as diverse as the design of integrated circuits, various scheduling problems, graph colouring and character recognition can be found in many papers (e g [22], [17], [34])

2.4 COMBINATORIAL OPTIMISATION

Many practical problems in a wide variety of areas concern themselves with the choice of a best configuration or set of parameters to achieve some goal Optimisation is devoted to solving such problems The *general optimisation problem* can be stated as follows [32]. Find x to

$$\begin{array}{ll} \text{minimise} & f(x) \\ \text{subject to} & g_i(x) \geq 0, \qquad i = 1, \quad ,m \\ & h_j(x) = 0, \qquad j = 1, \quad ,p \end{array}$$

where f, g_i, and h_j are general functions of the parameter $x \in R^n$ The widely known *linear programming problem* is the special case when f and all the g_i and h_j are linear functions

Optimisation problems may have either continuous or discrete variables in x *Combinatorial Optimisation* is the study of optimisation on a set of *discrete* variables, generally involving a search from a finite, often very large, set On the other hand, continuous optimisation is generally concerned with finding a set of real numbers Methods for solving continuous and combinatorial problems tend to be quite different

Some problems that do not appear discrete at first glance can be considered as combinatorial if solving them can be reduced to the selection of a solution from a finite set A notable example is linear programming Although variables are defined as continuous, the problem can be reduced as in the simplex method to the selection from a finite set We will see in Chapter 3 that our network design problem can be similarly reduced to a combinatorial problem

Combinatorial problems are usually very difficult to solve because of the *combinatorial explosion*, the tendency for problems to increase factorially in size with the number of variables Well-studied "benchmark" combinatorial problems include the travelling salesman problem, graph partitioning, graph colouring, bin packing, and a variety of routing and scheduling problems

Chapter 3. Complexity Issues: Reformulation as a Combinatorial Optimisation Problem

In Chapter 2, a number of variants on the long term topological design problem from the literature were discussed These modelisations differ in the cost function model employed We are concerned with the following four link cost functions (shown graphically in Chapter 2)

Case(a): Discrete function

The cost of having capacity c_l on link l is of the form

$$f_l^{(a)}(c_l) = \begin{cases} 0, & \text{if } c_l = 0, \\ k_{l_1}, & \text{if } 0 < c_l < c_{l_1}, \\ k_{l_2}, & \text{if } c_{l_1} \le c_l < c_{l_2}, \\ . \end{cases} \qquad (3\ 1)$$

where $k_{l_i} > 0$ and $k_{l_{i+1}} > k_{l_i}, \forall i$

Case(b): Concave function

Here, the cost of having capacity c_l on link l is given by a concave (nondecreasing) function, $f_l^{(b)}(c_l)$ with $f_l^{(b)}(0) = 0$

Case(c): Concave function with fixed part

$$f_l^{(c)}(c_l) = \begin{cases} f_l^{(b)}(c_l) + K_l, & \text{if } c_l > 0, \\ 0, & \text{otherwise}, \end{cases} \qquad (3\ 2)$$

where $f_l^{(b)}(c_l)$ is the concave nondecreasing function of (b) with $f_l^{(b)}(0) = 0$

Case(d): Linear with fixed cost case (the fixed charge problem)

Here, the cost of having capacity c_l on link l is

$$f_l^{(d)}(c_l) = \begin{cases} k_l c_l + K_l, & \text{if } c_l > 0, \\ 0, & \text{otherwise} \end{cases} \qquad (3\ 3)$$

In this chapter, we look at the various link cost functions under consideration and, for each one, we investigate the complexity of the overall optimisation problem We find, as expected, that the problem is most difficult when the cost function most closely

represents reality We would like to work with a cost function that is realistic and a problem that we can make a reasonable attempt to solve

A natural and convenient decomposition of the problem under consideration is presented in Section 3 1 and the subproblems identified are looked at separately for each cost function in Sections 3.2 - 3 4 Conclusions are drawn in Section 3 5 on the adoption of a suitable model for our work.

3.1 DECOMPOSITION OF THE PROBLEM

We can assess the relative difficulty of alternative cost functions by looking at what is involved in the optimisation problem if they hold Generally we are faced with two problems·

(i) What is the link configuration that produces an optimal solution?

(ii) What is the optimal routing, i e , given a link configuration how should capacities be installed to satisfy the demands?

It was noted at the end of Chapter 2 that the solution space, S, can be viewed as discrete if we can have optimal capacity assignment with only one path being used for each node pair Thus we may have a third consideration

(iii) Will the optimal routing involve *splitting* of flow?

Finding the answer to question (i) is computationally difficult and there are a large number of possible configurations for any problem of reasonable size Answering question (i) requires the answer to question (ii), as assessing how good a configuration is requires finding the total cost, which depends on the capacity assignments. The level of difficulty of (i) thus depends on how difficult it is to find an optimal routing, which in turn requires an answer to question (iii) - does splitting occur? We now look at these issues in reverse order

3.2 THE QUESTION OF SPLITTING

3.2.1 Discrete link cost function

It is sufficient to construct a simple example where splitting *is* necessary for optimal routing

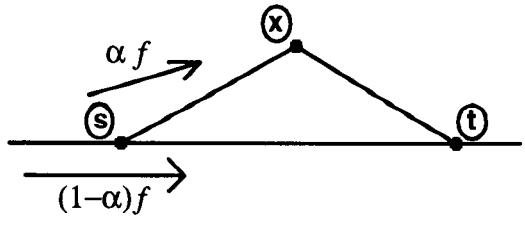


Figure 3.1

Assume the following

(i) We want to route f units of flow between s and t (Figure 3 1) Let flow αf be routed on the path s-x-t and $(1-\alpha)f$ on the path s-t ($\alpha \in [0,1]$)

(ii) Links (s,x), (x,t) and (s,t) already have some demands routed on them Each has capacity c_0 due to other flow requirements and can take $f/2$ units without cost increase (Figure 3 2) Any further increase would involve a jump in cost

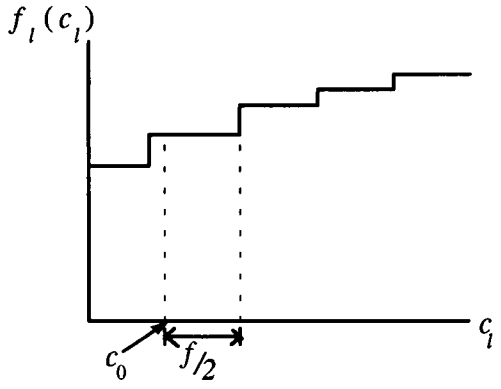


Figure 3.2 *Link cost function for links (s,x), (x,t) and (s,t)*

Then the cost attributable to the combined links of the two paths will have a minimum at $\alpha = ½$ Splitting of flow is thus necessary for optimal routing in this example We conclude that optimal routing may require flow splitting in this case

3.2.2 Concave link cost function

In this case, the cost of having capacity c_l on link l is given by a concave (nondecreasing) function, $f_l^{(b)}(c_l)$ with $f_l^{(b)}(0) = 0$

We wish to show that the required flow between any pair of nodes follows one and only one path for optimal routing Given required flow, f, between nodes s and t, and two alternative paths, p_1 and p_2, route αf via path p_1 and $(1-\alpha)f$ via p_2 (Figure 3 3)

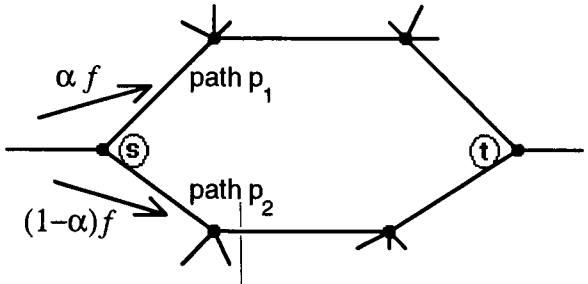


Figure 3.3

The cost attributable to the combined links of the two paths is

$$C_{(b)}(\alpha) = \sum_{l \in p_1} f_l^{(b)}(\alpha f + \Phi_l) + \sum_{l \in p_2} f_l^{(b)}((1-\alpha) f + \Phi_l) \quad , \quad \alpha \in [0,1]$$
$$= g_1(\alpha) + g_2(\alpha) \tag{3 4}$$

where Φ_l is the capacity on link l due to other flow requirements

To show that no splitting occurs, we must show that $C_{(b)}(\alpha)$ always has its minimum on the range [0,1] at one of the extremities, i e at $\alpha = 0$ or $\alpha = 1$ Thus it is sufficient to prove that its second derivative, $C''_{(b)}(\alpha)$, is always negative on [0,1], i e that there are no local minima

The first part, $g_1(\alpha)$, is a sum of positive concave functions, which is itself concave A nondecreasing concave function of α increases with α, but the rate of increase falls with α (Figure 3 4)

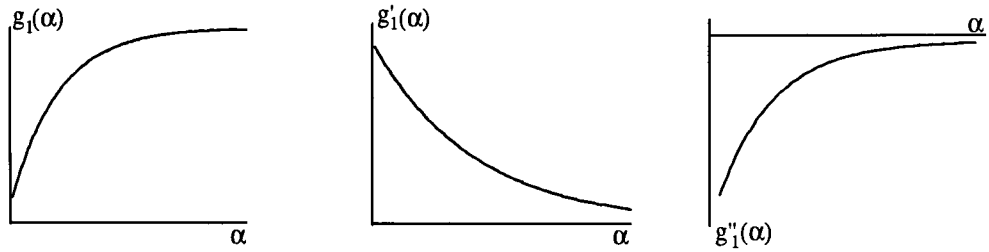


Figure 3.4 *A nondecreasing concave function and its first two derivatives*

The second part, $g_2(\alpha)$, is a reflection of $g_1(\alpha)$ in the line $\alpha = ½$ (Figure 3 5). This function decreases with α, and the rate of decrease also falls with α

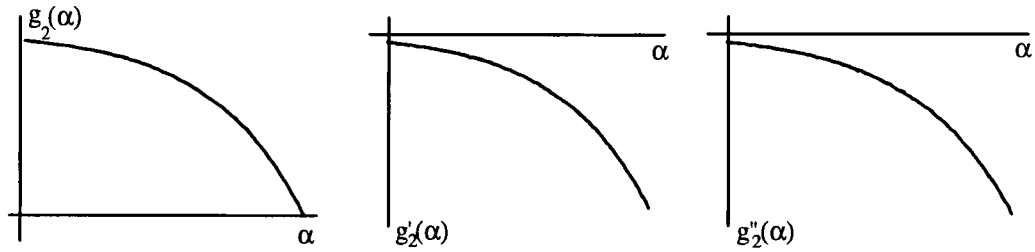


Figure 3.5 *A nonincreasing concave function and its first two derivatives*

It is clear from the above that

$$C''_{(b)}(\alpha) < 0 \qquad \forall \ \alpha \in [0,1], \tag{3 5}$$

as $g_1(\alpha)$ and $g_2(\alpha)$ are both negative on [0,1]

We conclude therefore that optimal routing does not require splitting for this case

3.2.3 Concave with fixed cost case

Here, the cost of having capacity c_l on link l is

$$f_l^{(c)}(c_l) = \begin{cases} f_l^{(b)}(c_l) + K_l & \text{if } c_l > 0, \\ 0 & \text{otherwise} \end{cases} \tag{3 6}$$

where $f_l^{(b)}(c_l)$ is the concave nondecreasing function of (b) with $f_l^{(b)}(0) = 0$

Equivalently,

$$f_l^{(c)}(c_l) = f_l^{(b)}(c_l) + K_l \ (c_l > 0) \tag{3 7}$$

Again, routing αf of flow f via path p_1 and $(1-\alpha)f$ via p_2 (Figure 3 3), we have

$$\begin{aligned} C_{(c)}(\alpha) &= \sum_{l \in p_1} f_l^{(c)}(\alpha f + \Phi_l) + \sum_{l \in p_2} f_l^{(c)}((1-\alpha)f + \Phi_l) \quad , \quad \alpha \in [0,1] \\ &= C_{(b)}(\alpha) + \sum_{l \in p_1} K_l \ (\alpha \neq 0) + \sum_{l \in p_2} K_l \ (\alpha \neq 1) \\ &= C_{(b)}(\alpha) + \text{const} \end{aligned} \tag{3 8}$$

We have shown that $C_{(b)}(\alpha)$ has its minimum on the range [0,1] at $\alpha = 0$ or $\alpha = 1$ We can see from (3 8) above that the same applies for $C_{(c)}(\alpha)$ and we can again conclude that optimal routing *is* possible without splitting of flow

3.2.4 Linear with fixed cost case

This is a special case of a concave function with fixed costs (Case (c) above), so optimal routing will not require splitting

3.3 FINDING AN OPTIMAL ROUTING

In our problem decomposition, subproblem (ii) was identified as given a link configuration, how should capacities be installed to satisfy the demands at minimum cost? With our decomposition, this subproblem is viewed separately from the broader problem of finding the optimal link arrangement It could be considered as a subroutine to be called for each link arrangement encountered by a solution algorithm

Where the link cost functions include a fixed initial charge (Cases (a), (c), (d)), we can say that the fixed cost has already been incurred if the link is present, and can thus be

ignored in the context of the routing problem Thus, given a link configuration, the routing problem is identical for cost functions (b) and (c)

For each demand, we will generally have a choice of paths on which to allocate capacity We showed in the previous section that one path is sufficient for optimal routing in cases (b), (c), (d) This path will be the 'shortest' path between the pair of nodes, where the 'length' of a link is the incremental cost of adding capacity onto that link

Case (a)

When link cost functions are discrete, optimal routing is very difficult as the link 'length' is not very well behaved Unlike the other cost functions we consider, optimal routing with a discrete cost function requires flow splitting The incremental cost of adding capacity will be zero most of the time, but will have sudden 'jumps'

Case(d)

When dealing with linear link cost functions (with or without fixed part), this incremental cost will remain the same throughout for a particular link Thus we can assign a constant length to each link[1]

The optimal path between a pair of nodes can be found by implementing a well-established shortest-path algorithm like that of Dijkstra (see Appendix 1 or any standard textbook on Graph Theory, e g [5][32]) When considering the allocation of capacities for a matrix of demands, it is inefficient to apply a single-pair shortest path algorithm separately for each pair It is best to use an all-shortest-paths algorithm, the most widely used being the Floyd-Warshall algorithm [5][32](Appendix 1) We will discuss these algorithms in more detail and introduce some efficiencies in their application to our problem in later chapters For the moment it is sufficient to state that the Floyd-Warshall algorithm is deterministic and can find the optimal routing in polynomial time, proportional to N^3, written $O(N^3)$, where N is the number of nodes Dijkstra's algorithm for a single shortest-path calculation takes $O(N^2)$ time

Cases (b) and (c)

Here, link 'length' will vary with capacity and the subproblem of finding an optimal routing is a difficult optimisation problem in its own right This subproblem has been

[1]It is noteworthy that these link lengths will correspond to the physical link lengths if there is a linear relationship between link cost and inter-node distance for a particular capacity This will often be the case and is the reason we receive link lengths as input for our problem

shown to be *NP*-hard[2] [16] A reasonably fast method for finding a *local* optimum was given by Yaged [35] in 1971 In [30], Minoux shows that the quality of the local optimum produced by this method is highly dependent on the starting solution chosen

3.4 Finding an Optimal Link Arrangement; A Combinatorial Optimisation Problem

The fact that link arrangements are discrete with a finite number of possibilities means that the subproblem of finding an optimal link arrangement is a combinatorial optimisation problem If it is possible to solve the routing subproblem in a straightforward manner, then the overall problem can be considered as a combinatorial problem When solving this combinatorial problem, a tractable subproblem would be presented for each potential solution considered

3.4.1 Complexity

The intrinsic difficulty in solving the routing problem for a discrete cost function makes the link arrangement problem extremely difficult in that case For link cost function (b), where fixed charges are not incurred, the optimal link arrangement problem *is* the optimal routing problem For the more realistic cost functions where a fixed charge is incurred, optimal link arrangement is time-consuming with the determination of an optimal routing required for each arrangement considered

In [30] Minoux shows that the *Optimum Network Problem*, shown to be *NP*-hard by Johnson, Lenstra & Rinnoy-Kan [18], is a special case of our optimisation problem where the link cost function is linear with fixed costs This proves that the more general fixed charge problem (d) is *NP*-hard As the routing subproblem is solvable in polynomial time, the link arrangement problem must be *NP*-hard for this case, the link arrangement problem for case (c) will likewise be *NP*-hard as it is, again, more general Proof of *NP*-hardness is useful as it justifies the development of approximation algorithms and time in not wasted in the search for an exact solution

3.5 Conclusion

The analysis in this chapter has shown that adopting a continuous approximation to the discrete link cost function makes the problem much more manageable.

---

[2]If a problem is *NP*-hard, it belongs to an important class of very difficult problems for which no polynomial solution has been found A full discussion on *NP*-hardness theory is given in [7]

Of the continuous functions, it is considered that those not involving a fixed link cost (i e (b)) are unrealistic Where real cost functions are considered in [20] by Kerner et al, fixed costs are large by comparison with capacity-dependent costs for the installation of both metallic and optical fibre cables

As the effective solution set is much smaller for problems with cost function (d) than (c), due to the straightforward routing problem, the prospect of working with this type of problem is more pleasant Also, it is felt that a concave (with fixed cost) approximation to the discrete cost function will be very nearly linear with only small economies of scale Thus we opt to concentrate on case (d), the fixed charge problem The *effective* solution space, mentioned above, will be the set of possible link arrangements (size of order 2^{N^2}) as the optimal routing problem has an exact solution

Chapter 4. Solution Methods

At the end of Chapter 3 we accepted the fixed charge problem as an appropriate model for our optimisation problem In this chapter we consider a variety of solution approaches, with particular emphasis on general rather than specific algorithms In Chapter 2 we established necessary constraints only, but noted that others could be required for some problem instances The solution methods considered are easily adaptable to such specific requirements

In Section 4 1, we investigate the use of greedy algorithms for our problem, with particular emphasis on Minoux's accelerated greedy algorithm Section 4 2 introduces local search as a class of approximation techniques Particular modern local search methods are investigated in the remaining sections, we will see that these methods have the appealing property that the algorithms do not become 'trapped' in local optima

4.1 GREEDY ALGORITHMS

Although our main interest is in the use of modern search methods for our problem, greedy algorithms merit further consideration for two reasons Firstly, as this type of algorithm has been most widely used in the past, it is a good benchmark by which new methods can be evaluated Secondly, it may be advantageous to incorporate a number of greedy iterations in an implementation of a broader search algorithm, this will become clearer when such algorithms are discussed in later sections

4.1.1 Standard Greedy Algorithm

In order to give a formal statement of Minoux's greedy algorithm, discussed in Chapter 2, in our terminology, we define $g(s^{(k)},l)$ as the minimum extra cost of re-routing all the flow which was running on link l on a single alternate path in the network given by solution $s^{(k)} \in S$ If no alternate path is available (if removing link l would cause non-connectedness), $g(s^{(k)},l)$ will be assigned a value of $+\infty$ We defined $f_l(c_l)$ for the cost function currently in question in Equation(3 3). The formal statement is as follows:

ALGORITHM 4.1: GREEDY ALGORITHM - MINOUX

(a) Select a starting solution, $s^{(0)} = \left(A^{(0)}, C^{(0)}\right)$ Let $k \leftarrow 0$

(b) At step k, let $s^{(k)}$ be the current solution

(i) $\forall\, l = (i, j) \in L$ such that $i < j$ and $a_l = 1$, compute[1]

$$\Delta^k(l) = g\left(s^{(k)}, l\right) - f_l(c_l) \tag{4 1}$$

(ii) Determine l' such that

$$\Delta^k(l') = \underset{\substack{l \in L \\ i<j}}{\text{Min}}\left\{\Delta^k(l)\right\} \tag{4 2}$$

(c) If $\Delta^k(l') \geq 0$, STOP Otherwise, let Λ^* be the set of links forming the path between the endpoints of l', obtained at (b)

(i) $\forall l \in L$, let $a_l^{k+1} \leftarrow a_l^k$ if $l \neq l'$, (4 3)

$a_l^{k+1} \leftarrow 0$ if $l = l'$

(ii) $\forall l \in L$, let $c_l^{k+1} \leftarrow c_l^k$ if $l \notin \Lambda^*$, (4 4)

$c_l^{k+1} \leftarrow c_l^k + c_l{}^k$ if $l \in \Lambda^*$

(iii) Set $k \leftarrow k+1$

Return to (b)

With the fixed charge cost function the cost, $g\left(s^{(k)}, l\right)$, can be calculated simply as the length of the shortest alternative path multiplied by the capacity, c_l, being re-routed

This greedy method has a significant drawback for our problem and modification is needed We wish to express a solution in terms of its link arrangement with only *optimal* capacity assignment possible for a particular arrangement The above algorithm may, however, provide solutions where capacity matrix, C, does *not* represent an optimal routing, given a link arrangement, A This is best illustrated by an example Consider the network of Figure 4 1 and assume that initially routing is optimal If link (2,4) is then removed, the best alternative path between its endpoints will be 2-3-4 Requirement (1,4) will have initially been routed 1-2-4 (shortest path) and will now be routed 1-2-3-4 This is sub-optimal, however, as the path 1-5-4 is shorter

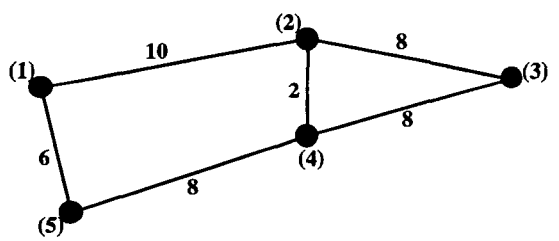


Figure 4.1 *Node index numbers are in parentheses Link lengths are shown beside links*

[1]We have constrained the adjacency matrix, A, to be symmetric so removal of link (i,j) implies removal of (j,i) as well

In order to maintain optimal routing, we need to look at all components making up the flow on a link when considering its removal All demands that are routed on the link must be re-routed on the best available path not containing that link Define $h\left(s^{(k)},l\right)$ as the extra cost involved when all requirements using link $l=(i,j)$ are re-routed optimally in the network given by solution $s^{(k)} \setminus \{(i,j)\}$ If removal of link l causes non-connectedness, or if link l is not present, we have $h\left(s^{(k)},l\right) \equiv +\infty$ Thus we can state our adaptation of Minoux's greedy algorithm

---

ALGORITHM 4.2 GREEDY ALGORITHM (ADAPTED)

As Algorithm 4 1 except

Replace equation (4 1) in step(b)(i) with

$$\Delta^k(l) = h\left(s^{(k)},l\right) - f_l(c_l) \tag{4 5}$$

Replace step (c)(ii) with the following

Update capacity matrix, C, so that routing in the new network is optimal

---

Minoux cites experimental evidence that solutions produced by his algorithm are almost always close to the global optimum (found by an exact method) We have no reason to fear that our adapted algorithm will not perform as well, the fact that routing is optimal at every iteration suggests that better performance may result Some comparisons have been done and are described in Chapter 6

4.1.2 Accelerated Greedy Algorithm

A major drawback of the greedy algorithm (especially with our adaptation) is that it is very slow Minoux shows that his algorithm has a worst-case complexity of $O\left(N^6\right)$ where the starting solution is a fully connected network. Our algorithm will be even slower with worst-case complexity of $O\left(N^7\right)$, this is due to the need to do extra shortest-path calculations at each iteration Recall Section 3.3, when it was stated that a standard single shortest-path algorithm takes $O\left(N^2\right)$ time and an all-shortest-path algorithm takes $O\left(N^3\right)$ time. This means that the solution of even medium-sized problems will be impractical with the greedy algorithm in its present form

Minoux presents an *accelerated greedy algorithm* to solve the problem more efficiently This algorithm will produce the same solution as the normal greedy algorithm if a condition of *monotonicity* holds, i e if, $\forall\, l \in L$,

$$\Delta^{k+1}(l) \geq \Delta^k(l), \tag{4 6}$$

where $\Delta \equiv +\infty$ when the link has been deleted The number of Δ computations can be greatly reduced when this condition holds In the first iteration, all Δ values are

calculated as before and the link corresponding to the smallest value (i e , largest reduction) is removed The link with the second smallest Δ value has it recomputed, if it is unchanged, or at least still smaller than the third smallest value, then it is removed as, by the monotonicity property, no other link can have a smaller Δ value If it is changed and its Δ value is no longer the second smallest, we proceed to the link with the next smallest Δ After this (second) link is removed, the process is repeated until no further improvement is possible With sizeable problems most Δ values will remain unchanged at each iteration and the number of computations is greatly reduced A formal statement of this algorithm (with optimal routing at each iteration) follows

---

ALGORITHM 4.3 ACCELERATED GREEDY ALGORITHM

(a) Select a starting solution, $s^{(0)} = \left(A^{(0)}, C^{(0)}\right)$ Let $k \leftarrow 0$

$\forall\, l = (i, j) \in L$ such that $i < j$ and $a_l = 1$, compute

$$\Delta(l) = h\left(s^{(0)}, l\right) - f_l(c_l), \tag{4 7}$$

where $h\left(s^{(0)}, l\right)$ and $f_l(c_l)$ are as defined earlier

(b) At the current step k, let $s^{(k)}$ be the current solution

(i) Determine l' such that

$$\Delta(l') = \min_{\substack{l \in L \\ i<j \;\; a_l \neq 0}} \{\Delta(l)\}, \tag{4 8}$$

(ii) Recompute $\Delta(l')$

If $\Delta(l')$ has been changed and

$$\Delta(l') \neq \min_{\substack{l \in L \\ i<j}} \{\Delta(l)\}, \tag{4 9}$$

then return to (b)(i)

(c) If $\Delta(l') \geq 0$, STOP

Otherwise

(i) Let $a_l^{k+1} \leftarrow a_l^k$ if $l \neq l'$,

$a_l^{k+1} \leftarrow 0$ if $l = l'$

(ii) Update capacity matrix, C, using shortest-paths algorithm

(iii) Set $k \leftarrow k+1$

Return to (b)

The reduction in complexity brought about by this acceleration is proportional to N^2 (Minoux) so, in the worst case, the algorithm will take $O\left(N^5\right)$ time In tests, however, the complexity is found to be somewhat less than this, even when a fully connected network is chosen as the starting solution

Minoux shows that the $\Delta^k(l)$ values of his greedy algorithm obey the monotonicity assumption and therefore the accelerated greedy algorithm always produces the same solution With our somewhat different re-routing strategy, this assumption does *not* hold Computational experiments (Chapter 6) show that, although the same result is found for many problem instances, some produce different results The results do not differ by very much, however, and sometimes the accelerated greedy produces a better solution than the standard greedy! Due to the much reduced complexity, Minoux [29] points out that 'for large problems, anyway, it is likely that only the accelerated greedy algorithm will be practicable'

4.1.3 An Example

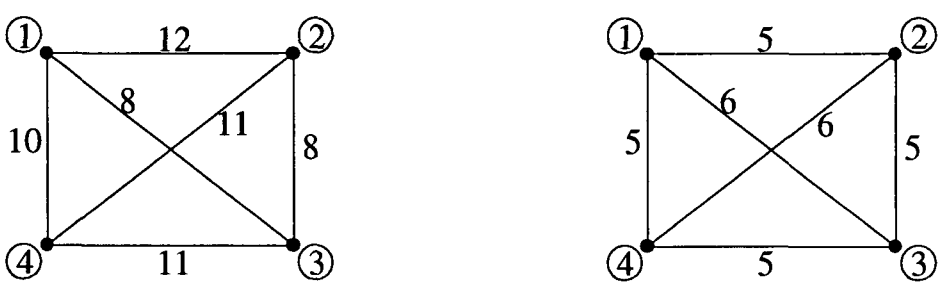
We will illustrate the implementation of the accelerated greedy algorithm for a very small example network Note that for clarity a symmetric example is chosen and all links may thus be considered bi-directional with all actions applicable in both directions

---

EXAMPLE 4.1 ACCELERATED GREEDY ALGORITHM

Given:

Input:



(a) Distances (b) Demands

Figure 4.2 *Input for our example*

Link Cost Function:

The standard form is

$$f_l(c_l) = K_l + k_l c_l \quad \text{if } c_l > 0, \qquad = 0 \quad \text{otherwise} \tag{4 10}$$

In this example, we will take both K_l and k_l as directly proportional to distance with constants of proportionality of 10 and 1 respectively Therefore, we have

$$f_{i,j}(c_{i,j}) = 10\, d_{i,j} + d_{i,j}\, c_{i,j} \quad \text{if } c_{i,j} > 0, \qquad = 0 \quad \text{otherwise} \tag{4 11}$$

Solution:

Starting solution:

We choose a feasible solution, $s^{(0)}$ to start, e g , a fully connected network Thus an optimal routing will involve direct routing of each demand over a single link in this case and the startmg solution will have the capacity allocations shown in Figure 4 3 (corresponding to the original demands) We have

$$\text{Total Cost} = \sum_{\text{all links}} f_l(c_l) = \mathbf{919}.$$

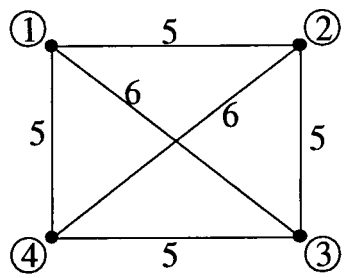


Figure 4.3 *Solution* $s^{(0)}$

Iteration 0:

By (4 7), we have

$$\Delta(1,2) = 80-180 = -100 \qquad \Delta(2,3) = 100-120 = -20$$
$$\Delta(1,3) = 120-128 = -8 \qquad \Delta(2,4) = 114-176 = -62$$
$$\Delta(1,4) = 95-150 = -55 \qquad \Delta(3,4) = 90-165 = -75$$

Minimum Δ value is for removal of link $l' = (1,2)$ Therefore, remove (1,2) As demand (1,2) only was using link (1,2), re-route on 1-3-2 and we have the capacity allocations of Figure 4 4, with a total cost of **819**.

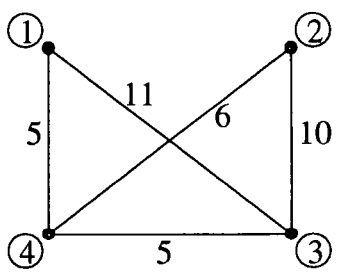


Figure 4.4 *Solution* $s^{(1)}$

Iteration 1:

As (1,2) is removed the next smallest Δ value is for $l' = (3,4)$ We recompute

$$\Delta(3,4) = 90-165 = -75,$$

and note that it is unchanged. Thus we choose to remove (3,4) As demand (3,4) only was using link (3,4), re-route on 3-1-4 and we have the capacity allocation of Figure 4 5, with a total cost of **744**

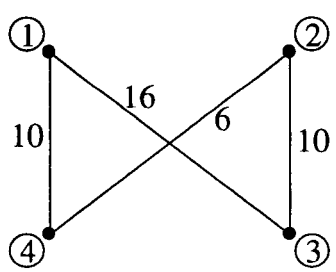


Figure 4.5 *Solution* $s^{(2)}$

Iteration 2:

As (1,2) and (3,4) are removed the next smallest Δ value is for $l' = (2,4)$ We recompute

$$\Delta(2,4) = -20,$$

and note that it has changed and is no longer the minimum Δ value The new minimum is for $l' = (1,4)$. Again, we recompute

$$\Delta(1,4) = -10$$

As *this* has also changed and is no longer the minimum, we go back to our Δ values and select the new minimum This turns out to be for l'=(2,4) Recomputing $\Delta(2,4)$ of course shows no change so we choose to remove this link In the current solution, $s^{(2)}$, link (2,4) is part of one path only, 2-4, so demand (2,4) is re-routed on 2-3-1-4 and we have the capacity allocation of Figure 4 7, with a total cost of **724**

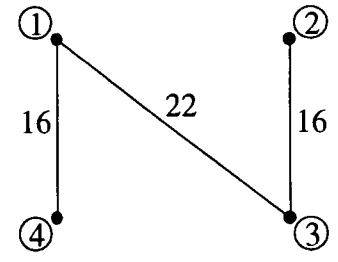


Figure 4.6 *Locally optimal solution* $s^{(3)}$

Iteration 3:

Although the three remaining links have recorded negative Δ values, these will become positive ($+\infty$ actually) when recomputed, so our current solution, $s^{(3)}$ is locally optimal

Note that this is not the global optimum, if link (1,4) is removed and link (3,4) inserted simultaneously, a solution with a cost of **702** could be obtained (Figure 4 7)

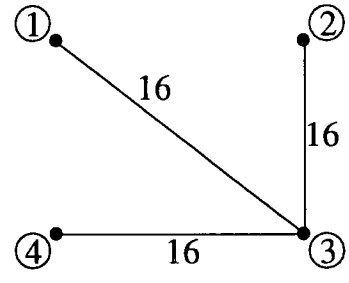


Figure 4.7 *Globally optimal solution (not obtained by the accelerated greedy algorithm)*

4.1.4 'Pure' Greedy algorithm

When link removal only is considered, choice of starting solution is critical as links not in this solution cannot form part of the local optimum found by any of the above algorithms The logical choice of starting solution, therefore, is a fully connected network A greedy algorithm where both link insertion and removal are considered at each iteration can be expected to be more robust as links not present in a starting solution can be added With link insertion, however, the monotonicity assumption is grossly violated and acceleration is not possible The results of an implementation of the standard greedy algorithm with link insertion as well as removal are given in Chapter 6 This algorithm is too slow, however, to merit serious consideration except in cases where only a few iterations are required We expect that solutions produced by search methods (e g simulated annealing) will be better than those of greedy algorithms with a fully connected or randomly chosen starting solution However, a 'good' solution, produced by one of these methods may be subject to slight improvement by

implementing a greedy algorithm using this result as a starting solution, in this case just a few iterations will be required and a better solution could be obtained by considering link insertion as well as removal

There are endless possibilities for the development of more elaborate greedy algorithms but a balance must always be struck between solution quality and time-complexity Frequently, the better solutions are provided by slower algorithms

4.2 LOCAL SEARCH METHODOLOGY

In this work, our primary interest is in the investigation of the use of recently developed methods in the solution of a particular optimisation problem. These methods, discussed in more detail in later sections, can be classified as local search approaches and thus a general introduction to local search methodology and related issues is useful

Local search (also called *neighbourhood search*) is a general approach to solving optimisation problems where there is a finite, but large, set of feasible solutions This approach is particularly suited to difficult combinatorial optimisation problems Many methods for solving continuous problems, however, like the simplex method for linear programming, can be said to have the same underlying principle The idea is that a *neighbourhood* is defined for any given solution as a set of solutions that are, in some sense, 'close' to the current one A new solution is selected from this set, the manner of selection is of course critical and describes a local search method The selection and adoption of a new solution is termed a *move* Local search methods are described in [17] as *metaheuristics* in that heuristics of particular problem types are 'fitted' to a general model This general applicability together with its foundation on the intuitive idea of perturbation make local search very appealing. The benefits of the ready adaptability of greedy algorithms were mentioned in Section 4 1, our greedy algorithms are local search methods

Formally, the local search approach can be stated as follows (a schematic representation is given in Figure 4 8)

ALGORITHM 4.4 GENERAL LOCAL SEARCH METHOD

(a) For a general solution, $s \in S$, define a neighbourhood, $N(s) \subset S$
Select a starting solution, $s^{(0)} \in S$ Let $k \leftarrow 0$

(b) Let $s^{(k)}$ be the current solution

Check if there exists an element of $N(s^{(k)})$ that is, in some sense, 'better' than $s^{(k)}$ If no such element exists, STOP

Otherwise Let $s^{(k+1)}$ be the 'best' choice from $N(s^{(k)})$, let $k \leftarrow k+1$, and return to (b)

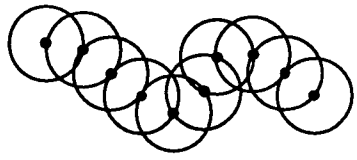


Figure 4.8 Schematic representation of local search, the neighbourhood of a point (solution) is the circle about it

---

With traditional local search approaches, only improving moves are accepted and a local optimum is reached when no further improving move can be found Modern approaches such as the methods of simulated annealing, tabu search and genetic algorithms differ in that they allow non-improving moves under certain conditions in order that better solutions can be found

Local search is described and general issues explored by Papadimitriou & Steiglitz in their 1982 book [32] and also in a paper by Ulrich Dorndorf [6] Issues of complexity are examined in papers by Johnson et al [19] and Yannakakis [36] and a class, *PLS* (*P*olynomial-time *L*ocal *S*earch), is defined to characterise local search complexity The issues of significance when implementing a local search approach are identified in [6] as

- how to choose an initial solution,
- how to define neighbourhoods, and
- how to select a neighbour of a given solution

Choice of an initial solution will depend of course on the local search method, for greedy-type algorithms this choice is critical, but where non-improvmg moves are allowed the choice of initial solution may not be so important In the former case it may be possible to make a judicious choice (fully connected network for the accelerated greedy algorithm, for example), alternatively, many runs with random starting solutions may be necessary

In the definition of neighbourhoods, a clear trade-off exists between quality of local optima obtained and algorithm complexity Larger neighbourhoods would mean that a better result is found, but at the expense of increased complexity, as searching the neighbourhood would become more time-consuming In [32], it is noted that the choice of neighbourhood structure is usually guided by intuition and experience as very little theory is available For our problem, the neighbourhood of a solution is defined as the set of solutions differing in link arrangement from the current one by one link Thus a move consists of a link insertion or removal It is desirable that a neighbourhood

structure possesses the *connectivity* property [6] so that any solution can be reached from any other in a finite sequence of moves The asymmetric[2] neighbourhood structure of our accelerated greedy algorithm (link removal only allowed) means that the connectivity property does not hold, causing the solution quality to be highly dependent on starting solution choice The neighbourhood structure could, of course, be made more complex and intelligent, whereby features common to good (or bad) solutions could be identified and then fixed, resulting in reduced neighbourhood size [32] It may also be found that a solution algorithm is fooled by adopting 'tempting' solutions on the basis of short term gain, these solutions could also be identified and removed from the neighbourhood set

The selection of a neighbour to replace the current solution is the third issue identified above Examination of the entire neighbourhood in order to select the best solution constitutes a greedy approach but this may be very time-consuming (especially if evaluating the cost of a given solution is time-consuming, as is the case with our problem formulation) and another (e g random) approach may be preferable Again, elaborate approaches where the selection process has memory and learns from experience may be useful This search approach is the essence of a general local search method

4.3 MODERN APPROACHES

We describe the principles of each of three new local search methods in this section, namely *simulated annealing, tabu search* and *genetic algorithms* These are among five promising methods, discussed by Glover & Greenberg in a 1989 review paper [13], which, although quite different from one another, have a common ability to avoid becoming trapped in local minima.

4.3.1 Simulated Annealing

The method of simulated annealing, introduced by Kirkpatrick, Gelatt & Vecchi in 1983 [22] and independently by Cerny in 1985, is based on a strong analogy between the physical process of annealing of solids and the problem of solving large combinatorial problems

[2]If a neighbourhood structure is *symmetric* then

$$s' \in N(s) \Leftrightarrow s \in N(s')$$

In the physical annealing process, the material is first melted and then *slowly* cooled until the particles arrange themselves in a state of minimum energy, the *ground state* In the melted state the particles of the material move freely with respect to one another, but this thermal mobility is lost with cooling Slow cooling is necessary in order that there is sufficient time for redistribution of these particles in an even way If the temperature is lowered too quickly, locally suboptimal configurations will be frozen into the material, causing it to end up in a non-homogeneous state of higher than the minimum energy

The mechanics of the annealing process can be efficiently modelled by the *Metropolis Algorithm* (1953). The cooling process is viewed as a sequence of discrete states through which the material progresses. Given the current state i with energy E_i, the next state j, with energy E_j, is generated by the application of a small distortion (e g particle displacement) to the system The change in energy,

$$\Delta E = E_j - E_i, \tag{4 12}$$

is computed If $\Delta E \leq 0$, the new state, j, is accepted as the current state If $\Delta E > 0$ the new state is accepted with probability p, given by the *Boltzmann distribution*

$$p = \exp\left(\frac{-\Delta E}{k_B T}\right), \tag{4 13}$$

where T represents the system temperature and k_B is a physical constant (Boltzmann's constant) At a given iteration when $\Delta E > 0$, acceptance will depend on a comparison between p and a random number uniformly distributed on $[0,1)$ At high temperatures where $k_B T >> \Delta E$, p will be close to 1 and most non-improving changes will be accepted As T is lowered and p approaches 0, however, less and less are accepted

In [22], an analogy with combinatorial optimisation is used to develop the simulated annealing method Feasible solutions of the optimisation problem are considered equivalent to states of a physical system The cost of a solution is equivalent to the energy of a state A *control parameter, c*, is introduced which has the same influence as temperature in the physical system. This is initially set at a 'high' value and lowered in some gradual way to a 'low' value, according to a *cooling schedule*, controlling the number and size of cost function increases accepted

We previously noted that a possible neighbourhood structure for our problem is such that a move corresponds to a link insertion or removal A neighbourhood would then be the set of solutions differing from the current one by the absence or presence of one link. An advantage of using this neighbourhood structure will become clear in Chapter 6 when we see that the time required for recalculating cost on a single link insertion or removal is significantly less than that for another (arbitrary) new solution

Adopting this structure, a general simulated annealing approach can be stated as follows.

ALGORITHM 4.5 SIMULATED ANNEALING

(a) Select a starting solution, $s^{(0)} = \left(A^{(0)}, C^{(0)}\right) \in S$ Let $k \leftarrow 0$

Let c_0, the initial value of the control parameter, be a 'high' value

(b) At step k, let $s^{(k)}$ be the current solution with cost $F\left(s^{(k)}\right)$ Let c_k be the current value of the control parameter

(i) Select a random solution, $s' = (A', C')$, from $N\left(s^k\right)$

i e select a link index number l and, $\begin{cases} \text{if } a'_l = 1, \text{ let } a'_l \leftarrow 0; \\ \text{if } a'_l = 0, \text{ let } a'_l \leftarrow 1 \end{cases}$

Calculate the cost, $F(s')$ and let $\Delta F = F(s') - F\left(s^{(k)}\right)$ (This requires finding C' by a shortest-path algorithm)

(ii) Let $p = \begin{cases} 1, & \text{if } \Delta F \le 0, \\ \exp\left(\dfrac{-\Delta F}{c_k}\right) & \text{if } \Delta F > 0 \end{cases}$

If $p >$ random$[0,1)$,

let $s^{(k+1)} \leftarrow s'$,

Otherwise,

let $s^{(k+1)} \leftarrow s^{(k)}$

(c) Update c according to cooling schedule

If stopping criterion is true, STOP

Otherwise let $k \leftarrow k+1$,

return to (b)

It has been proven that the simulated annealing algorithm converges to a globally optimal solution, but that the cooling schedule required takes infinite time! Determination of a suitable finite-time cooling schedule requires insight and/or trial-and-error experiments for the algorithm to be effective We describe the implementation of such a cooling schedule, consistent with [1], in Chapter 5 together with results in Chapter 6

4.3.2 Tabu Search

Tabu Search is another newly developed general method for solving difficult combinatorial problems, and is based on the idea of intelligent problem-solving where

the solution process has a degree of *memory* This method was introduced in its modern form by Fred Glover in 1986 and expanded in a two-part series of papers in 1989 and 1990 [11][12] A comprehensive development of Tabu Search theory is given in [15] together with a review of many applications and a discussion of connections with other procedures and the potential for creating hybrid approaches combining elements of these methods Some experience of the implementation of Tabu Search is also presented in [17]

Tabu Search is essentially a deterministic process with the emphasis on more systematic forms of guidance, in contrast to simulated annealing which uses randomisation as a method of escaping local optima

Traditional local search methods move from solution to solution as long as the move causes a favourable change in the objective function When no further improving move is possible, a local optimum is said to be reached, and the process terminates An extension of this approach would be to accept the move causing least cost function deterioration in the event that no improving moves are possible It is very likely, however, that the process will revert to the local optimum on the next move and the only effect of our extension will be to introduce infinite cycling close to a local optimum If, however, reverse moves are temporarily classified as forbidden (tabu), this tendency to oscillate can be overcome and the process can be guided out of local minima

This is *recency-based* Tabu Search in its simplest form and memory is short-term Oscillations could still occur, however, with longer 'period' than the list length so some element of medium- or long-term memory may be necessary There is much scope for establishing very elaborate memory structures to suit both general and specific problems In implementing a tabu search strategy, optimisation subproblems arise, e g how long should the tabu list be? Details of our implementation are given in Chapter 5, reasonable memory structures are selected for use with reference to our initial experiments and those discussed in the papers referred to above A general tabu search algorithm is stated below

---

ALGORITHM 4.6 TABU SEARCH

For a general solution, $s \in S$, define a neighbourhood, $N(s) \subset S$ Define a tabu list, T Given a solution, s, define $T(s)$ as the subset of $N(s)$ which is tabu as a result of certain moves being in T

(a) Initialise $T = \varnothing$ (tabu list)
Select a starting solution, $s^{(0)} \in S$ Let $k \leftarrow 0$
Let $\hat{s} = s^{(0)}$ (best solution so far)

(b) Let $s^{(k)}$ be the current solution

If $N(s^{(k)}) - T(s^{(k)}) = \varnothing$ STOP (i e if the entire neighbourhood is tabu)

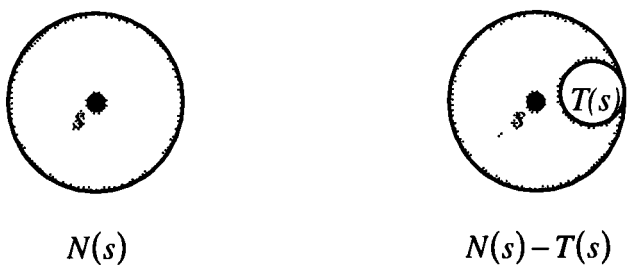
Otherwise Let $s^{(k+1)}$ be the *best*[3] choice from the set $N(s^{(k)}) - T(s^{(k)})$,

Update T,

Let $k \leftarrow k+1$,

Return to (b)

Thus Tabu Search strategically reduces the size of the neighbourhood to be explored This is demonstrated schematically in Figure 4 9



$N(s)$ $N(s) - T(s)$

(a) Neighbourhood structure for traditional local search

(b) Neighbourhood structure for Tabu Search

Figure 4.9 *Schematic representation of the reduced neighbourhood structure with Tabu Search*

4.3.3 Genetic Algorithms

Genetic algorithms, introduced by John Holland in 1975, are based on mechanisms of natural selection that are believed to apply in genetics In contrast to other methods, breadth of performance is achieved by working with a *population* of solutions New solutions are propagated from old ones in a randomised fashion, with the 'fitter' solutions more likely to survive The average quality of solutions is improved from generation to generation by the *exchange of information*

Genetic algorithms work with a coding of a solution rather than the solution itself This coding is usually a *string* of 1's and 0's Each solution of our combinatorial optimisation problem, the link arrangement subproblem, is represented by an adjacency matrix This matrix can be easily transformed into a string as it also contains just 1's and 0's

A stage in the execution of a genetic algorithm is called a *generation* The first generation usually consists of a set of randomly chosen strings (solutions) A new

---

[3]Solution corresponding to move of greatest gain or smallest loss

generation is 'bred' from the current one by the execution of the following three component processes

(i) Reproduction,

(ii) Crossover,

(iii) Mutation

Reproduction is the means by which strings are selected according to their relative costs ('fitness' values) to contribute offspring for the next generation Strings with higher fitness values will have a higher probability of being selected. This is analogous with Darwin's *survival of the fittest* theory

The set of strings selected by reproduction is said to constitute the *mating pool Crossover* is the procedure whereby pairs of individuals from the mating pool exchange information, with the prospect of fitter strings being occasionally created In practice this means that some binary bits of one string are swapped with the corresponding bits of another The fittest of these new solutions will tend to be retained through future generations by the reproduction process, while those that prove to be unfit will be discarded

Mutation is simply the introduction of a small random element into a genetic algorithm It is realised by the random inversion of a bit in a string The probability of mutation is usually set to a low value so that there will be about, say, one mutation per generation Mutation is necessary because it accommodates reversing the loss of some potentially important genetic information in the reproduction and crossover processes Also, as a genetic algorithm progresses, all the individuals in the population of strings tend to converge to one string (effectively a local optimum) and many string positions will have the same value for all individuals Mutation is useful for giving the search increased diversity as no amount of reproduction and crossover alone will change these values

This is the procedure in its most straightforward form (the Simple Genetic Algorithm), a more detailed discussion of the algorithm and many extensions is given in [34] A hybrid approach that is considered to be particularly suited to our problem is presented in Chapter 5 together with results in Chapter 6

4.4 TABULAR COMPARISON OF METHODS

A simplified comparison of the four approaches considered is given in Table 4 1

	Greedy Algs	Sim Annealing	Tabu Search	Genetic Algs
Local Search Method?	Yes	Yes	Yes	Yes
Number of solutions carried from iteration to iteration	1	1	1	>1
Probabilistic/ Deterministic	Det	Prob	Det	Prob
Level of dependence on initial solution chosen?	High	Low	Low	Low

Table 4.1

Chapter 5. Implementation

Having described the principles of the solution methods that interest us, we now concern ourselves with their adaptation and use for solving our problems Our algorithms were initially implemented in their simplest forms The purpose of this initial implementation was to get hands-on experience of the advantages and limitations of the methods so that potential further improvements could be evaluated more critically The final implementations chosen draw on this experience

General implementation issues are considered in Section 5 1 The efficient use of shortest-path algorithms, central to all implementations, is discussed in Section 5 2 A mechanism for generating realistic test problems is described in Section 5 3 Section 5 4 discusses in general the implementation of the solution methods. The remaining sections contain detail on the adaptation and implementation of the individual solution methods under consideration, as follows

5 5 Greedy Algorithms
5 6 Tabu Search
5 7 Simulated Annealing
5 8 Genetic Algorithms

5.1 GENERAL ISSUES

In our implementations, algorithms are coded in the 'C' programming language Editing, testing and debugging of the software was performed on a personal computer and on Sun SPARC workstations A SPARC 10 station was used for performing the optimisations Portable code was written for all implementations (i e code performing to the ANSI C standard) General issues considered significant to the implementation of solutions to our type of problem are considered here

5.1.1 Data Structures and Memory Considerations

Input variables

Recall that we are considering the fixed charge problem as a working approximation to reality We are also assuming that the total installation cost can be expressed as a sum

of link costs The cost of installing link l has been expressed in terms of capacity c_l, as

$$f_l(c_l) = \begin{cases} K_l + k_l c_l, & \text{if } c_l > 0, \\ 0, & \text{otherwise} \end{cases} \qquad (5\ 1)$$

Thus, to calculate cost at each iteration, matrix representations of K_l, the fixed link costs, and k_l, the variable (capacity-dependent) costs, are required The demand matrix, R, introduced in Chapter 2, must also be set up There is no need to set up a separate distance matrix, however, as the only concept of 'distance' of interest to us for a particular link is provided by the variable cost matrix (which we expect will be closely related to distance) It is quite possible that we will receive information at this stage of the design process that will require 'translation' into our form We may, for example, be given a distance matrix and proportionality constants relating fixed and variable cost elements to distance

Output variables

We have defined the required output in terms of an adjacency matrix, A, and a capacity matrix, C These $(N \times N)$ matrices must also be initialised

Temporary variables

We must be able to represent the paths established between node pairs in a given solution in an efficient way

With a cost function of our type (or indeed a general concave cost function), we can state

(i) Optimal routing can be achieved with just one path for each inter-node demand (Section 3 2)

(ii) If the optimal path between a pair of nodes (i, j) is the sequence of nodes, $i = i_0, i_1, \quad, i_n = j$, then the optimal path between a node pair (i_p, i_q), $0 \le p < q \le n$, is a subsection of this path This is proven by Yaged [35]

Thus the following recursive definition is unambiguous

Define

$$p(i, j) = k, \qquad (5\ 2)$$

where k is the node immediately preceding j on the path from i to j A single $N \times N$ matrix, $P = (p_{i,j})_{i,j=1,\ ,N}$, is sufficient to represent all paths in a given solution

The path from i to j is found in the following recursive manner

(a) Let the route be

$$i_0, i_1, \quad, i_n,$$

Initially,

$i_0 = i$ and $i_n = j$ are known,

i_1, , i_{n-1} and n are unknown

Let $k = 0$

(b) Repeat

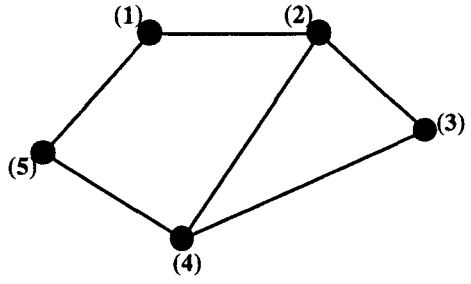
let $k \leftarrow k+1$,

let $i_{n-k} = p(i, i_{n-k+1})$,

until $i_{n-k} = i_0$

Let $n = k$

A simple example may help to clarify this Suppose we have the network and path matrix of Figure 5 1



P	1	2	3	4	5
1	-	1	2	2	1
2	2	-	2	2	1
3	2	3	-	3	1
4	2	2	4	-	4
5	5	1	2	5	-

Figure 5.1 *Network and path matrix for our example*

We wish to read the path between, say, nodes 3 and 5 from the path matrix Firstly, we find that $p(3,5) = 1$, so the path will be 3– –1–5 Now $p(3,1) = 2$ so we update the path, 3– –2–1–5 This is the last step required as $p(3,2) = 3$ so we conclude that the path is 3–2–1–5

Other paths are found similarly as follows

1-2	2-1	3-2-1	4-2-1	5-1
1-2-3	2-3	3-2	4-3-2	5-1-2
1-2-4	2-4	3-4	4-3	5-1-2-3
1-5	2-1-5	3-2-1-5	4-5	5-4

Memory

We would like our algorithms to be applicable to problems where N is of order 100 The scale of memory on widely available computers suggests that quite a few $N \times N$ matrices could be set up with N of this order For some of the methods under consideration and their extensions, however, much extra memory is required to store a

number of solutions (e g genetic algorithms) or other information (e g tabu search), and when N is large the scale of memory required may exceed that available It is worthwhile, therefore, to develop a coding strategy to reduce redundancy in the data structures Such a coding strategy is used for storing tabu search information in our implementation

5.1.2 Random Number Generator

The effectiveness of both simulated annealing and genetic algorithms is dependent on the quality of the random number generator (RNG) used A RNG is also required for the generation of quasi-random test problems A good quality random number generator is unbiased over its range and has a large period The Wichmann-Hill algorithm is used in our implementation

5.2 SHORTEST PATH ALGORITHMS

We showed in Section 3 3 that, for the case of a linear cost function with fixed part, the optimal routing for a particular link arrangement is found by finding the 'shortest' path between each node pair. The 'length' of a link is the incremental cost of adding capacity onto that link, this is constant for our case All the solution methods under consideration involve moving from solution to solution in some manner and, for each solution considered, an optimal routing must be performed to calculate the cost of that solution This calculation of shortest paths is very time-consuming and occupies most of the computing time for all of our implementations

We are thus motivated to give some thought to efficient implementation of shortest path algorithms

5.2.1 Single pair shortest path algorithms

A straightforward and very efficient algorithm for finding the shortest directed path from any node to any other node is given and described in Appendix 1 This was first proposed by Dijkstra in 1959, and it takes $O(N^2)$ running time

5.2.2 All shortest paths

A straightforward and widely used algorithm for finding the shortest paths between all pairs of nodes is also given in Appendix 1 This algorithm was proposed by Floyd and Warshall (separately) in 1962 and takes $O(N^3)$ time

We could use this algorithm to establish inter-node paths for the initial solution, and again after every iteration to update the paths and maintain optimal routing. This would be very slow, however; we propose an improved procedure, applicable to our problem, in Section 5.2.4.

5.2.3 Improvements in efficiency - literature

Very many iterative procedures (including all that we are looking at) require time-consuming shortest path calculations at every iteration. This has motivated considerable effort in the area and a large body of work has been published. Improved algorithms have been produced by exploiting the features of particular types of networks; decomposition algorithms have, for example, been developed for sparse networks. Reference is made in [14] to an annotated bibliography of 448 articles that appeared before 1982 on shortest paths! A shorter annotated bibliography of the most significant work is given by Lawler in [24]. It is beyond the scope of this project to perform a detailed study of shortest path algorithms apart from those for general networks described in the above sections; we have, however, achieved considerable efficiencies in our procedures by adopting an approach for *updating* shortest paths, as presented in the next subsection.

5.2.4 Improvements in efficiency - Updating all shortest paths

If a move is defined for our problem as a link insertion or removal (or even a few insertions or removals), a significant improvement in efficiency can be achieved by taking the approach developed here. Clearly, for a network of any significant size, there is much duplication of previous work in the recalculation of all shortest paths after a move; here information from the current configuration is used in the establishment of the next one. We look at link insertion and removal separately.

Link Insertion

If a link (p,q) is inserted, we assume that (q,p) is inserted simultaneously; this is consistent with our earlier assumptions (Section 2.3) that the adjacency matrix remains symmetric.

The idea here is that any change in the shortest path from i to j after the insertion of links (p,q) and (q,p) will have (p,q) or (q,p) as part of the new path. Thus, if we define $\lambda_{i,j}$ as the length of the shortest path between i and j in the direction $i \rightarrow j$, $\forall i, j \in V = \{1,\cdots,N\}$, we have the following procedure:

ALGORITHM 5.1 LINK INSERTION

$\forall i, j \in V$,

Find the minimum of

(i) λ_{ij}

(ii) $\lambda_{ip}+d_{pq}+\lambda_{qj}$

(iii) $\lambda_{iq}+d_{qp}+\lambda_{pj}$

and record which of (i), (ii), (iii) gives this minimum

If the minimum is given by (i),

Path between i and j remains unchanged,

Else if (ii) is the minimum,

Update $\lambda_{ij}=\lambda_{ip}+d_{pq}+\lambda_{qj}$,

New shortest path is given by merging the (current) path from i to p, link (p,q), and the path from q to j,

Else if (iii) is the minimum,

Update $\lambda_{ij}=\lambda_{iq}+d_{qp}+\lambda_{pj}$,

New shortest path is given by merging the (current) path from i to q, link (q,p), and the path from p to j,

Next i,j

This algorithm requires $O(N^2)$ time, as there are $N(N-1)$ node pairs to consider

Link Removal

It has been shown (Section 4 1 1) that simply re-routing the flow on link (p,q) on a new shortest path between p and q can result in non-optimal routing. An alternative procedure is to ascertain which inter-node paths contain the link (p,q) and re-route these on the shortest paths in the network with (p,q) removed

We must compare the computational complexity of this procedure with that of a new all-shortest-paths calculation If there are k routes using link (p,q), k single-pair shortest path calculations are required, so the time-complexity is $O(k\ N^2)$ Clearly, our new approach is only beneficial if the complexity is not greater than that of the Floyd-Warshall algorithm, $O(N^3)$, thus we must compare k to N

The maximum value that k can take is found by partitioning the set of nodes in the network into two subsets, as shown in Figure 5 2

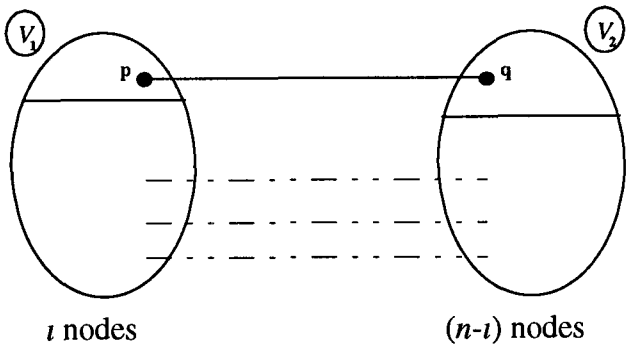


Figure 5.2 *Partitioning of node set, V into* $\{V_1 \quad V_2\}$

Some fraction of the nodes in V_1, say $\alpha\,\iota$, will be connected to some fraction of those in V_2, say $\beta\,(n-\iota)$, via (p,q) Then $\alpha\,\iota\,\beta\,(n-\iota)$ routes will contain (p,q). $\iota(n-\iota)$ has a maximum of $N^2/4$ at $\iota = N/2$ Thus the maximum possible value for k is $N^2/4$, when $\alpha = \beta = 1$ and $\iota = N/2$ The minimum value for k is of course 0

We conclude that we should perform k Dijkstra calculations only if $k < \hat{k}$, and otherwise use an all-shortest-path algorithm, where $\hat{k}$ is of the same order as N Tests have shown that choosing $\hat{k} = N$ is about optimal Thus we have derived a link insertion procedure that can occasionally take $O(N^3)$ time, but frequently takes only $O(N^2)$ time and is thus much faster than the simple approach of doing a full re-calculation of all the shortest paths at every iteration

It is worth pointing out that we will usually have $k < \hat{k}$ when the current solution graph is well connected k becomes larger on average as the graph becomes more sparse because we can expect more routes to use a given link This explains differences in runtime where the same algorithm is used for problems with the same number of nodes Progression will be slower where the solutions encountered are mostly sparsely connected

In our implementation, we create a new data structure, *routes_using*[*p*][*q*], to keep a dynamic record of the value of k for each potential link removal, so that the decision (is $k < \hat{k}$?) can be made quickly The set of links using (p,q) is not readily available with the data structures already defined, but can be retrieved from a recursive search of the path matrix (defined in Section 5 1 1) In theory, of course, this set of links could be stored directly in memory for each node pair, but practical limits on memory space would make this impossible for any sizeable N If stored in array form, an array of order $N \times N \times N^2$ would be required

Conclusion

The average time taken for a move involving a single link insertion or removal has been reduced as the routing will very often take time proportional to N^2.

5.3 GENERATION OF TEST PROBLEMS

It is important to ensure that our evaluations are independent of the idiosyncrasies of any particular network. We propose a mechanism for generating random, realistic test networks in this section. As we are concerning ourselves with the fixed charge problem, a classification of fixed charge problems is introduced to examine how algorithm performance depends on the nature of the problem. This classification is defined by a parameter that gives a measure of the relative significance of initial (fixed) link costs. Random networks with specific properties can then be generated.

5.3.1 Classification of Fixed Charge Problems

The fixed charge problem corresponds to having the following cost function (equation 3.3):

$$f_l(c_l) = \begin{cases} K_l + k_l c_l, & \text{if } c_l > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

Define a *characteristic number* k_{char} for a fixed charge problem as follows:

$$k_{\text{char}} = \frac{F_{\text{fixed}}}{F_{\text{var}}}, \tag{5.4}$$

where

$$F_{\text{fixed}} = \sum_{l=(i,j)\in G^*} K_l, \tag{5.5}$$

and

$$F_{\text{var}} = \sum_{l=(i,j)\in G^*} k_l c_l, \tag{5.6}$$

where G^* is the optimal solution. This number is a measure of the ratio between fixed and variable costs in the optimal solution.

We can classify problems according to this characteristic number k_{char}. We consider in particular the following three cases:

(a) The fixed cost is much larger than the variable cost, i.e. $k_{\text{char}} >> 1$; (say $k_{\text{char}} = 10$).
(b) The variable cost is much larger than the fixed cost, i.e. $k_{\text{char}} << 1$; (say $k_{\text{char}} = 0.1$).
(c) The fixed and variable costs contribute about equally to total cost, i.e. $k_{\text{char}} \approx 1$.

We would like to be able to generate problems with a given k_{char} but we have the problem that it is determined from the optimal solution (which we do not know in advance!) Thus the above definition is not very useful and we define $\bar{k}_{char}$ as an approximation to k_{char} as follows

$$\bar{k}_{char} = \frac{\bar{F}_{fixed}}{\bar{F}_{var}}, \tag{5 7}$$

where

$$\bar{F}_{fixed} = \sum_{l=(i,j)\in L_b} K_l, \tag{5 8}$$

and

$$\bar{F}_{var} = \sum_{l=(i,j)\in L_b} k_l c_l, \tag{5 9}$$

where c_l are capacities corresponding to the minimal solution of the all-shortest-paths problem on the base graph, G_b Recall that L_b represents the set of links of the base graph These capacities $\{c_l\}$ can be easily found, given a set of demands and a base graph

5.3.2 Generation of Test Problems

Here we wish to create test problems from scratch We firstly specify the number of nodes, N, and the base graph The N nodes are placed in a user-defined 2-dimensional space A matrix of distances between the node pairs is set up by taking the Euclidean distance between the nodes and applying a random distortion This distortion is applied because link length does not correspond to Euclidean distance in reality It is convenient to define two new variables for each link, f_l^{fixed} and f_l^{var}, such that

$$\begin{aligned} K_l &= f_l^{fixed} d_l, \\ k_l &= f_l^{var} d_l \end{aligned} \tag{5 10}$$

These variables relate link fixed and variable costs respectively to distance We can expect f_l^{var} to be fairly constant, but f_l^{fixed} will vary considerably They are distributed evenly on reasonable values The demands are also generated randomly Finally, problems with a desired characteristic number can be generated by firstly generating an intermediate problem, finding its characteristic number, and scaling appropriately Our generation mechanism thus consists of the following seven steps

(i) Select the number of nodes, N

(ii) Let $l=(i,j)$ be an element of L_b, the links of the base graph, with probability $\xi \in [0,1]$ ξ is denoted the *base graph connectedness* In general, it may be

desirable to impose restrictions on the form of this base graph It must for example be connected We also impose a restriction in our test problems that the base graph remain connected after the removal of any one link (i e it must be *2-edge connected*) This ensures that single-edge connectivity is not dependent on any particular link being present This restriction is realised by imposing a 2-edge connectivity test and repeating step (ii) on failure

(iii) Let the co-ordinates (x_i, y_i) of each node i be uniformly distributed on a 2-dimensional space, $[0, S_1] \times [0, S_2]$ Calculate elements of the distance matrix as

$$d_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}\ \varepsilon, \tag{5 11}$$

where ε is a uniform random variable on $[a,b]$

(iv) Let demands $r_{i,j}$ have a uniform distribution on $[0, r_{\max}]$

(v) $\forall l = (i,j) \in V$, let each f_l^{fixed} and f_l^{var} have uniform distributions on $[\alpha_{\text{fixed}}, \beta_{\text{fixed}}]$ and $[\alpha_{\text{var}}, \beta_{\text{var}}]$ respectively

(vi) Find $\bar{k}_{\text{char}}$ for this problem If $\bar{k}'_{\text{char}}$ is the desired characteristic number, then scale the problem by applying a constant factor to each of the fixed link costs, as follows $\forall l \in L_b$, let

$$f_l^{\text{fixed}\,'} = f_l^{\text{fixed}} \frac{\bar{k}'_{\text{char}}}{\bar{k}_{\text{char}}} \tag{5 12}$$

5.3.3 Parameter choices

It was decided to fix some of the parameters at the outset All test problems used have the following (realistic) settings

$$r_{\max} = 10,\ S_1 = S_2 = 100,\ a = 1,\ b = 1\,5,$$
$$\alpha_{\text{fixed}} = 1,\ \beta_{\text{fixed}} = 2,\ \alpha_{\text{var}} = 1,\ \beta_{\text{var}} = 1\,5$$

A set of six different 20-node test problems, denoted PROB1, ,PROB6, were generated for most of our tests These are over a variation of values of $\bar{k}_{\text{char}}$ and ξ, as follows

$\bar{k}_{\text{char}}$	$\xi = 0\,8$ (dense)	$\xi = 0\,3$ (sparse)
0 1	PROB1	PROB4
1 0	PROB2	PROB5
10 0	PROB3	PROB6

Table 5.1 *Test problem set*

Thus, for example, PROB1 has relatively low fixed costs and a densely connected basic graph

Other test problems were also used and are discussed later All were generated using the above procedure

5.4 IMPLEMENTATION OF SOLUTION STRATEGIES

As suggested earlier, a 'natural' neighbourhood structure for local search implementations is such that a move involves a single link insertion or removal Our implementations, described here, take this approach and have much in common Thus software has been developed to perform the following common functionality for all our implementations

- Read input data
- Write / Display output
- Select initial solution(s) and perform initial path and link capacity assignments (Floyd - Warshall algorithm)
- Perform a move - update paths and capacities for link insertion or removal, using the improved algorithm of Section 5 2 4
- Generate random numbers

Issues relating to the above have been discussed in Section 5 1 and 5 2, implementations of the particular strategies are described in the following sections

5.5 GREEDY ALGORITHMS

The use of greedy algorithms to solve this kind of problem was discussed in Chapter 4 Algorithms developed for this study draw on Minoux's work [30] and extend it further The following four variations were tested on a number of problems

(i) Minoux's standard greedy algorithm (see Section 4 1 1)
(ii) Adaptation of (i) to our model (Section 4 1 1)
(iii) Accelerated greedy algorithm (Sec 4 1 2)
(iv) 'Pure' greedy algorithm (Sec 4 1 4)

Recall that the first three variants permit link removal only Variant (iv) is pure in the sense that it is a true, symmetric local search method, allowing moves to be undone occasionally It will be seen that our Tabu Search implementation builds on this

The algorithms, as formally stated in Section 4.1, were coded in C. Care was taken to achieve the best possible efficiency in the implementation of sorting, etc. Initial solutions are set up in a random fashion with the desired level of connectedness. If a full initial solution is desired (i.e. 100% connected), there will be of course be no randomness as there is just one such solution to choose from.

A variety of results and comparisons are given for tests on these algorithms in the next chapter.

5.6 TABU SEARCH

We introduced the fundamental ideas of Tabu Search in Section 4.3.2 by showing how memory could be used to guide a greedy search process out of local optima. Clearly this sort of implementation of Tabu Search will be useful as a direct comparisons can then be made with greedy algorithms.

Thus our basic implementation involves a process which initially is just like greedy algorithm (iv) (link removal *and insertion* must of course be permissible). The difference, however, is that the process does not terminate on reaching a local optimum. Where no improving moves are possible, the best non-improving move is chosen.

5.6.1 Short-term memory

A tabu list, L_1, is created to prevent cycling close to local optima. As cycling is caused by the reversal of previous moves, recording the links corresponding to these moves and prohibiting operations on these links realises a tabu list. L_1 must be limited in length for three reasons:

(i) Potential moves must be checked against each list element before being performed. This is time consuming.
(ii) The list occupies valuable memory space.
(iii) If a large portion of the potential moves (i.e. the neighbourhood) becomes tabu, the process can be restricted excessively in its search for good solutions.

A list of a fixed maximum length is realised by just remembering recent moves and 'forgetting' the least recent list entry when a new move is added. The list then resembles a sliding window. Memory space is reserved for a fixed number of moves with each new one overwriting the least recent. Our experiments show that best results (in terms

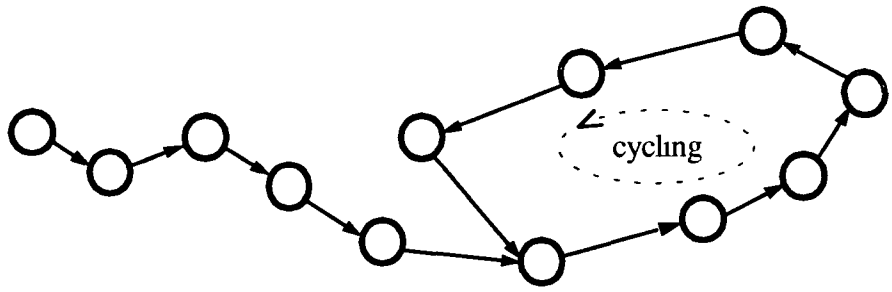


Figure 5.3 Demonstration of cycling

of solution quality and time) are achieved for tabu lists of a size between 5 and 20 This is consistent with Glover's perhaps remarkable observations [15] that tabu lists of this size are best for almost all applications!

5.6.2 Longer-term memory

The method presented so far is purely deterministic This means that if a solution is reached more than once, cycling has occurred (Figure 5 3), and will continue for ever

Short-term memory prevents immediate cycling, but a cycle of length greater than the tabu list can still occur Long term memory is necessary for such cycles to be broken, they should be broken at the earliest possible time as nothing is gained when going over old ground

Cycling can only be prevented by recognising previously encountered states Keeping an exhaustive list of previous states is however similar to having a very long tabu list and is undesirable The solution is some 'sampling' of previous states and we could, for example, record every 50th state and thus build up a list containing states 50, 100, 150, 200, . Every new state encountered could then be checked against this list This is somewhat unsatisfactory, however, as the list would get very long for implementations requiring a large number of iterations (each iteration is a state), say thousands Problems of time (to check new states against a long list of states) would arise as well as memory space problems To overcome this in our implementation, we take the approach of having a *non-linear* sampling of states

The idea is to take an integer $n > 1$ ($n = 4$ is chosen for our implementation) Then set up a list, L_2, with k elements to contain the following at each iteration i

$$L_2 = [\text{state } x_1 n \text{ where } x_1 n < i \le (x_1 + 1)n,$$
$$\text{state } x_2 n \text{ where } x_2 n^2 < i \le (x_2 + 1)n^2,$$
$$\text{state } x_k n \text{ where } x_k n^k < i \le (x_k + 1)n^k \ ]$$

Thus with $n = 4$, we will have the following list L_2 at $i = 510$ (for example):

L_2 =[state 508, state 496, state 448, state 256, state 0, ..., state 0]
$(127*4)$ $(31*4^2)$ $(7*4^3)$ $(1*4^4)$ $(0*4^5)$ $(0*4^k)$

This list is realised by checking at each iteration whether the iteration's index number is a multiple of 4, 16, 64, 256, 1024, etc., and updating the relevant element(s) of L_2 appropriately. All multiples of 16 are of course also multiples of 4, all multiples of 64 are multiples of 16 and 4, and so on.

Table 5.2 clarifies this further by showing the states in L_2 for the first 21 iterations of a Tabu Search process with a choice of $n = 4$.

iteration	States in L_2	iteration	States in L_2	iteration	States in L_2
1	[0,0,0,...]	8	[8,0,0,...]	15	[12, 0, 0,...]
2	[0,0,0,...]	9	[8,0,0,...]	16	[16,16,0,...]
3	[0,0,0,...]	10	[8,0,0,...]	17	[16,16,0,...]
4	[4,0,0,...]	11	[8,0,0,...]	18	[16,16,0,...]
5	[4,0,0,...]	12	[12,0,0,...]	19	[16,16,0,...]
6	[4,0,0,...]	13	[12,0,0,...]	20	[20,16,0,...]
7	[4,0,0,...]	14	[12,0,0,...]	21	[20,16,0,...]

Table 5.2 *21 iterations of a tabu search process with* $n = 4$

Three issues arise with this approach.

(i) Choice of n and k. The larger n is the smaller the list length k needs to be to detect cycles of a given maximum size. When n is large, however, performance deteriorates as cycling is not so quickly detectable. Our tests show choice of $n = 4$ to be about optimal. With $n = 4$, choosing $k = 6$ allows detection of cycles up to 4096 states in length, which is more than sufficient for our application. Thus we just need to keep a record of six states at any time.

(ii) Storage of a state. A state in the process is a solution of our network topological design problem together with the short-term tabu list L_1. A solution is represented by its adjacency matrix and L_1 by a fixed-length list of links. It would be cumbersome to store all the information in this form for each element of L_2 as very many comparisons would be required to check each new state against the list. A new state might only differ from a state in L_2 by one link. A more efficient strategy is adopted whereby each state is coded as an integer. It would be possible to have a unique integer for each state, but this is not really necessary. The coding strategy adopted is such that the probability is very small that two different states being compared will have the same code.

(iii) How is cycling overcome when it does arise? Intuitively, the best strategy is to apply a small perturbation to the system to break the cycle In our case, we alter (insert or remove) a small number of links, chosen at random, and reset the tabu list An alternative is to restart the entire process, choosing a different random starting solution Initial tests have shown that the latter (perhaps surprisingly) produces better results. This is because a broader search is performed when a larger number of runs are done, and the benefit of this outweighs that of longer individual searches

Figure 5 4 demonstrates this schematically Note that the cycle may not be broken immediately, a few iterations will be required (depending on n and the length of the cycle).

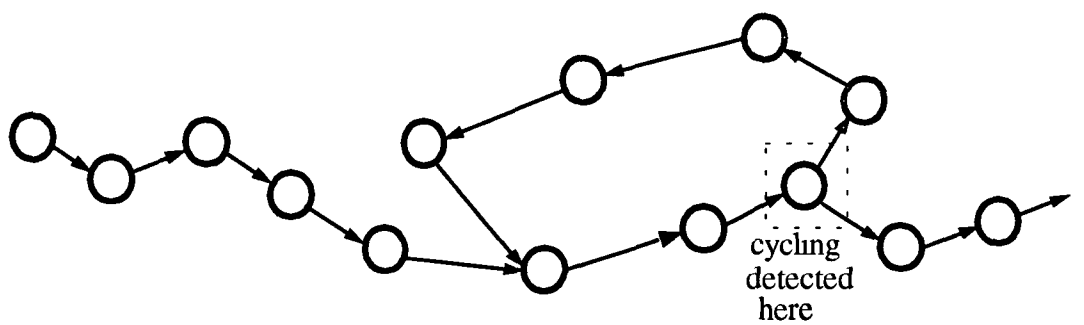


Figure 5.4 *Escape from cycling*

A variety of runs were performed with different starting solutions (including a fully connected network), results are presented and comparisons made in the next chapter

5.7 SIMULATED ANNEALING

The implementation of a Simulated Annealing algorithm is perhaps the most straightforward among those under consideration, as very little information needs to be carried from iteration to iteration The basis of this method was described in Section 4 3 1 and its application to our problem was described in a general way

In our implementation, the basic idea is that at any iteration, k, a random link is selected from the set of all *possible* links If that link is present in the current configuration, the cost of its removal is calculated, if it is absent the cost of its insertion is calculated The new configuration is accepted with probability p, defined by:

$$\begin{aligned} p &= 1 && \text{if } \Delta C \leq 0, \\ &= \exp(-\Delta C / c_k) && \text{if } \Delta C > 0, \end{aligned} \qquad (5\ 13)$$

where ΔC is the change in total cost resulting from the move.

The major issue in implementing a simulated annealing algorithm is the determination of a suitable cooling schedule

5.7.1 Cooling Schedule

It was noted in Section 4 3 1 that the simulated annealing algorithm has been proven to converge to a globally optimal solution, but that the cooling schedule required would take infinite time We require a *finite-time approximation*, that is a practical cooling schedule, that can achieve a good solution (hopefully globally optimal or close to it)

In our implementation, the form of the cooling schedule is consistent with that described in [1] and is as follows

(i) The initial value of the cooling parameter, c_0, is chosen so that virtually all transitions are accepted at first A tentative (small) value of c_0 is chosen initially and a number of iterations performed If the proportion of potential moves accepted is less than a certain value (close to 1), c_0 is multiplied by a constant factor and the process repeated In our implementation we firstly set c_0 to a fairly low value and perform N^2 iterations (where N is the number of nodes, a measure of the size of the problem), recording the number accepted If the acceptance ratio is less than 99%, c_0 is doubled and the process repeated A value for c_0 is finally accepted when it is sufficiently large that the acceptance ratio exceeds 99%

(ii) After a number of transitions, n, the parameter is reduced exponentially, according to

$$c_{k+1} = \alpha\, c_k, \qquad (5\ 14)$$

where α is smaller than but close to 1. Best results were consistently achieved in our experiments for α between 0 9 and 0 99

(iii) The algorithm is terminated when new random configurations are very rarely accepted or when c becomes very small (i e when c reaches c_{min}) There is no point in continuing the algorithm if the chances of achieving further improvement are very slim

(iv) An upper bound, n_{max}, is placed on the number of attempted transitions per step before the cooling parameter is reduced This is because transitions are accepted with decreasing probability and step lengths get very large as c is lowered Given values of α between 0.9 and 0 99 (like in our implementation), values of n and n_{max} are chosen so that the algorithm will stop after a specified time

A simplified pictorial representation of the behaviour of the cooling parameter c_k, is given in Figure 5 5

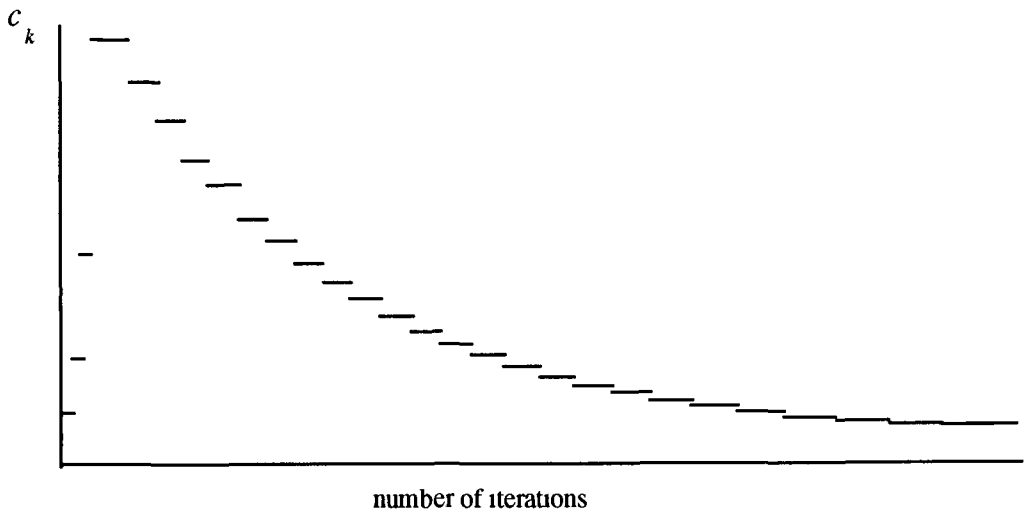


Figure 5.5 *Typical behaviour of cooling parameter* c_k

5.8 GENETIC ALGORITHMS

5.8.1 The Simple Genetic Algorithm

The principles of the simple genetic algorithm were introduced in Section 4.3.3. A population of solutions is selected to be the first state and new solutions are *propagated* from this state by means of reproduction, crossover and mutation.

This algorithm, with a number of enhancements, was applied to the fixed charge problem by a colleague at Dublin City University. This work and results are presented in [31]. The most important parameters to be set are identified as

- population size,
- probability of crossover, and,
- probability of mutation.

A variety of runs were performed with variations of these parameters, bearing in mind the ranges recommended in the literature. Two major problems in the use of pure genetic algorithms for our problem were identified as

(i) Carrying a population of (say 100) solutions from iteration to iteration is computationally very expensive. At each iteration, all the shortest paths would need to be updated for *each* of the solutions. Recall that most of the computation time in all our implementations is in the calculation of shortest paths. Furthermore, the change in a given solution on an iteration may be in more than one link (if crossover occurs), adding to the time required for the shortest paths update.

(ii) Although moderately good solutions are obtained in reasonable times, very much more time is required for convergence to the global optimum (or close to it) This is because all solutions tend to converge prematurely towards the same solution, and further improvement is just achieved by infrequent mutations

Problem (ii) can be solved reasonably well by implementing a greedy algorithm after the initial convergence to a fairly good solution A few greedy iterations would then bring about further rapid convergence and the process could be ended The algorithm still takes too long, however, due to the reason specified in (i) above, and a new approach is needed In the next subsection, we describe a new algorithm that uses the principles of genetic algorithms, but that has much better performance.

5.8.2 An adaptation of genetic algorithms to our problem

Initial tests have shown quick convergence of greedy algorithms to (usually good) local optima. The following approach attempts to combine the principle of exchange of information among a number of solutions with this rapid convergence The idea is to select, as the population of solutions, local optima produced by different runs of a greedy algorithm Information is exchanged between these solutions and the new solutions are used for further greedy iterations Information is again exchanged between the resulting local optima, and the process repeated It has been noted that carrying a significant number of solutions is computationally very expensive Because of this, and to simplify matters, we restrict ourselves to a population of two solutions

Heuristic

Thus we proceed as follows

Select two different (random) starting graphs, $G_1^{(0)}$ and $G_2^{(0)}$.
Perform a greedy algorithm on each, giving local optima $\overline{G}_1^{(0)}$ and $\overline{G}_2^{(0)}$
Exchange information (by crossover and mutation), to get $G_1^{(1)}$ and $G_2^{(1)}$
Perform greedy algorithms on each, to get $\overline{G}_1^{(1)}$ and $\overline{G}_2^{(1)}$.

Stop after a certain number of iterations or on convergence of G_1 and G_2 to the same graphs

Figure 5 6 gives a graphical description of this process

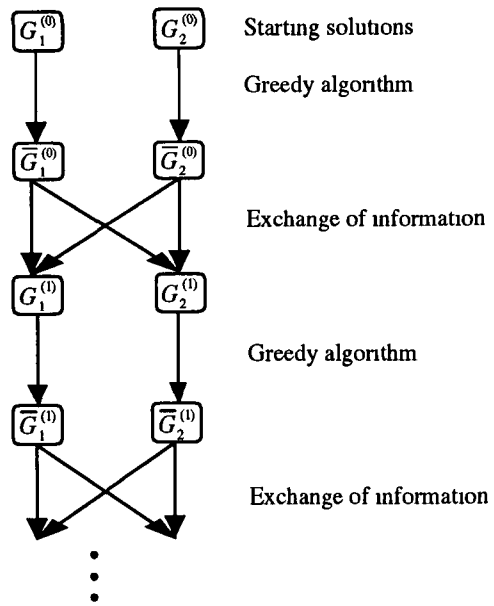


Figure 5.6 *Schematic demonstration of a greedy/ genetic algorithm hybrid*

Exchange of Information

Information between graphs G_1 and G_2 is exchanged using the following mechanism

Partition the set of all links L_b in the base graph into 4 non-overlapping subsets,

$$L_b = \{L_1 \mid L_2 \mid L_3 \mid L_4\}$$

where the L_i's represent the following

L_1 set of links *not* present in G_1 or G_2,

L_2 set of links in G_1 only,

L_3 set of links in G_2 only,

L_4 set of links in both G_1 and G_2

Define a probability of mutation, p_{mut}

Define the following crossover probabilities

$p_{\text{add}1}$, $p_{\text{add}2}$ probability of adding links to G_1 and G_2 respectively by crossover

$p_{\text{remove}1}$, $p_{\text{remove}2}$ probability of removing links from G_1 and G_2 respectively by crossover

$\forall l \in L_1$,

add link l to G_1 with probability p_{mut} (mutation)

add link l to G_2 with probability p_{mut} (mutation)

$\forall l \in L_2$,

remove link l from G_1 with probability $p_{\text{remove}1}$ (crossover)

add link l to G_2 with probability $p_{\text{add}2}$ (crossover)

$\forall l \in L_3$,

add link l to G_1 with probability $p_{\text{add}1}$ (crossover)

remove link l from G_2 with probability p_{remove2} (crossover)

$\forall l \in L_4$,

remove link l from G_1 with probability p_{mut} (mutation)

remove link l from G_2 with probability p_{mut} (mutation)

Results are presented for this heuristic in the next chapter

5.9 CONCLUSION

This chapter has been concerned with the practical use for solving our problem of a variety of solution methods As all these methods are applicable to a large range of optimisation problems, each adaptation required some quantity of new work Specific guidelines were followed where appropriate

The method chosen for solving the shortest-paths subproblem (i e the method of *updating* all shortest paths) presented in Section 5 2 4 is novel and was developed specifically for this work

The greedy algorithms specified in Section 5 5 draw on Minoux's work [30]; the extensions described here and in Chapter 4 are new work, however.

The implementation of Tabu Search short-term memory is quite straightforward and is the same as that in most of the relevant literature (e g [15], [17]) On the other hand, the particular implementation of longer-term memory chosen is novel

Our simulated annealing implementation (Section 5 7) follows closely the guidelines set out in [1]

The simple genetic algorithm implementation discussed is based almost entirely on the work of Dave O'Meara in [31] The hybrid genetic/greedy algorithm presented in Section 5 8 2 is however new work, developed in conjunction with Dmitri Botvich

Finally, the technique used for the generation of test problems (Section 5 3) was also developed specifically for this project in conjunction with Dr Botvich

Chapter 6. Results and Comparisons

This chapter is organised as follows. Following an introduction, there are four sections that are concerned with individual results for each of the general optimisation procedures under consideration, namely,

- Greedy Algorithms,
- Tabu Search,
- Simulated Annealing, and,
- Genetic Algorithms.

In Chapter 5, we described heuristic adaptations of these approaches to our topological network design problem. These descriptions introduced a variety of parameters to be set for each implementation. The best values for these parameters can often be found simply by trial-and-error; the results presented in Sections 6.2 to 6.5 help us to draw conclusions on the best parameter choices. Some of the 'best' implementations of each approach are then compared with each other in an overall comparison in Section 6.6.

6.1 INTRODUCTION

It is important to be clear on what we are looking for from our results. The following three criteria are considered most important in evaluating the algorithms.

(i) Cost; Our objective function is a representation of monetary cost and this must be minimised.

(ii) Robustness; How well-behaved is the algorithm? An algorithm that produces exceptionally low cost solutions in some circumstances may not rank as highly overall as one which does reasonably well all the time.

(iii) Speed; As computation time is limited, the length of time required for convergence is very significant.

It is useful at this stage to reproduce Table 5.1 to recall the six 20-node test problems that are used in many of our tests. Note that the problem generator was configured with the realistic parameter set given in Section 5.3.3.

$\bar{k}_{char}$	$\xi = 0.8$ (dense)	$\xi = 0.3$ (sparse)
0.1	PROB1	PROB4
1.0	PROB2	PROB5
10.0	PROB3	PROB6

Table 6.1: *Test problem set*

To ensure that comparisons are valid, all optimisations are performed on the same machine. Times measured are in seconds of CPU time on a SPARC 10. Actual times would be much longer (say 5 or 10 times) on a slower machine or in a multi-processor environment.

6.2 COMPARISON OF GREEDY ALGORITHMS

6.2.1 Comparison of results for greedy algorithms

In this section we are interested in seeing how our implementation of the Accelerated Greedy Algorithm compares with Minoux's Normal Greedy Algorithm (adapted to our model). It is also interesting to observe the performance of a greedy algorithm with link insertion as well as removal permitted as this is the basis of our Tabu Search approach.

Tables 6.2 (i)-(vi) display results for each of these algorithms over our six 20-node test problems. It is important to note that just relative, as opposed to absolute, cost values are meaningful. Comparisons between absolute costs across the problem set are also meaningless.

As well as convergence times, we are interested in the performance of these algorithms over a range of fixed charge problems and over runs with different starting solutions. Each run was performed with a random initial solution as well as a solution with all possible links in place (the base graph).

PROB 1		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		0.5	166 345		0.6	217 526
Std. Greedy		12.8	166 345		3.5	217 526
Gr with ins.		18.4	166 345		43.9	166 345

(i) k_{char} =0.1, ξ =0.8

PROB 2		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		1.1	193 196		0.6	243 762
Std. Greedy		28.2	193 196		10.1	243 401
Gr with ins.		41.9	193 196		39.0	193 955

(ii) k_{char} =1.0, ξ =0.8

PROB 3		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		2.0	298 687		1.2	373 887
Std. Greedy		35.8	299 328		15.3	373 887
Gr with ins.		54.6	299 328		31.1	305 362

(iii) k_{char} =10.0, ξ =0.8

PROB 4		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		0.6	291 346		0.5	420 126
Std. Greedy		4.7	291 346		0.9	420 126
Gr with ins.		7.4	291 346		18.6	291 346

(iv) k_{char} =0.1, ξ =0.3

PROB 5		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		1.0	394 059		0.6	529 888
Std. Greedy		10.9	394 059		2.2	529 888
Gr with ins.		17.2	394 059		16.3	394 560

(v) k_{char} =1.0, ξ =0.3

PROB 6		Full Initial Solution			Random Initial Solution	
Algorithm		time (s)	Cost		time (s)	Cost
Acc. Greedy		1.5	922 938		0.9	1 293 905
Std. Greedy		13.8	935 775		3.7	1 293 905
Gr with ins.		22.7	935 775		8.8	1 088 054

(vi) k_{char} =10.0, ξ =0.3

Table 6.2 *Results for various greedy algorithms over 6 fixed charge problems*

Observations:

(i) The Accelerated Greedy Algorithm is by far the fastest in all cases

(ii) All three algorithms are highly dependent on the initial solution chosen, which is a cause for concern with their use. Fortunately, however, the results found are very good if a full graph is chosen as the starting solution As would be expected, this dependence on starting solution is less for the third case where link insertion as well as removal is permitted

(iii) Finally, it was observed in chapter 4 that the *monotonicity criterion* behind the accelerated greedy algorithm is necessarily violated occasionally in our implementation This criterion is a condition for the Accelerated Greedy Algorithm to produce identical results to the Standard Greedy (link removal only) We see in Table 6 1 that the same result is found for most problem instances but that there are occasional differences It is reassuring to see in our results that this violation is more likely to bring about an improvement rather than a deterioration in performance.

It was noted in (i) above that the Accelerated Greedy Algorithm has the best time-performance The next sub-section shows that this is even more marked as the problem complexity increases

6.2.2 Performance improvement with the Accelerated Greedy Algorithm

Figure 6 1 displays a comparison of how much time is required for the accelerated greedy algorithm with that for the standard greedy algorithm for a variety of test problems with different numbers of nodes These problems are all generated with the same generation parameter settings (except the number of nodes) Each has a characteristic number $k_0 = 1\ 0$ In each case the basic graph contains about 80% of possible links

The dramatic improvement achieved by the accelerated greedy algorithm confirms our expectations of a reduction in complexity proportional to N^2, where N is the number of nodes

6.2.3 Conclusion

Notwithstanding some concerns about its robustness, the Accelerated Greedy Algorithm is most worthy of serious consideration, primarily for reasons of computational efficiency

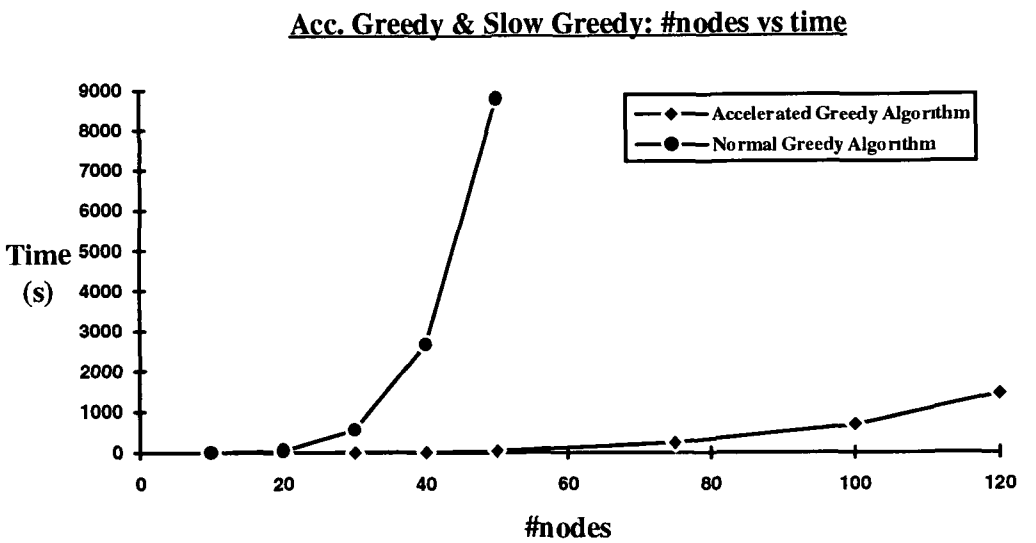


Figure 6.1: *Time performance of standard and accelerated greedy algorithms for 'average' homogeneous input with* $k_{char} = 1\,0$ *and* $\xi = 0\,8$

6.3 SOME TABU SEARCH STRATEGIES

In this section, we wish to examine some variants of the Tabu Search implementation discussed in Chapter 5 Recall that we implement a short-term memory by having a sliding window of tabu moves Best results have been seen for tabu lists of size 5 to 20 (consistent with Glover's observations) The algorithm presented in Section 5 6 2 implements longer term memory and prevents infinite cycling without using excessive amounts of memory Parameters of the algorithm are set as follows $n = 4, k = 6$

Tables 6 3 (i)-(vi) display results for each of the following four algorithms over our six 20-node test problems As before, each run was performed with a random initial solution as well as a solution with all possible links in place

Algorithm	Tabu list size	No. of iterations
Tabu Search 1	5	1 000
Tabu Search 2	5	10 000
Tabu Search 3	20	1 000
Tabu Search 4	20	10 000

Time:

1000 iterations takes approximately 550 seconds on average 10000 iterations takes about ten times as long (~1 5 hours)

PROB 1	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	166 345	166 345
Tabu Search 2	10 000	166 345	166 345
Tabu Search 3	1 000	166 345	166 345
Tabu Search 4	10 000	166 345	166 345

PROB 2	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	193 196	193 196
Tabu Search 2	10 000	193 196	193 196
Tabu Search 3	1 000	193 196	193 196
Tabu Search 4	10 000	193 196	193 196

PROB 3	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	297 542	296 613
Tabu Search 2	10 000	295 600	295 600
Tabu Search 3	1 000	297 995	296 912
Tabu Search 4	10 000	296 409	296 912

PROB 4	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	291 346	291 346
Tabu Search 2	10 000	291 346	291 346
Tabu Search 3	1 000	291 346	291 346
Tabu Search 4	10 000	291 346	291 346

PROB 5	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	393 992	393 992
Tabu Search 2	10 000	393 992	393 992
Tabu Search 3	1 000	394 059	394 560
Tabu Search 4	10 000	393 992	394 560

PROB 6	#iterations	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Tabu Search 1	1 000	921 331	921 331
Tabu Search 2	10 000	921 331	921 331
Tabu Search 3	1 000	935 775	948 532
Tabu Search 4	10 000	931 575	928 358

Table 6.3 *Results for some variations on our Tabu Search implementation*

Figures 6 2 - 6 4 below give a graphical feel for the progression of the Tabu Search algorithm for one thousand iterations on a particular problem

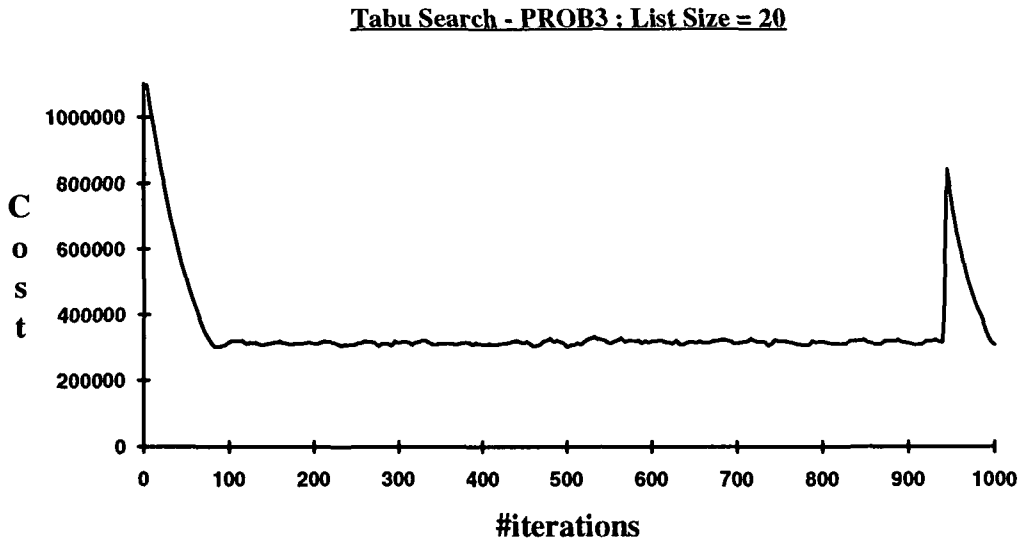


Figure 6.2 *Progression of "Tabu Search 3" for problem "PROB 3" - list size = 20*

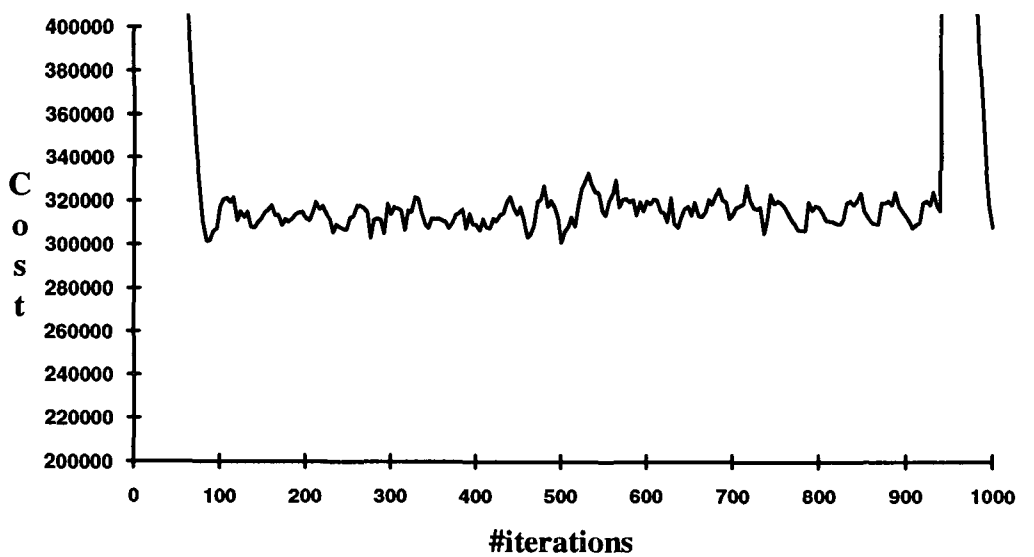


Figure 6.3 *"Close-up" of the above (Figure 6 2)*

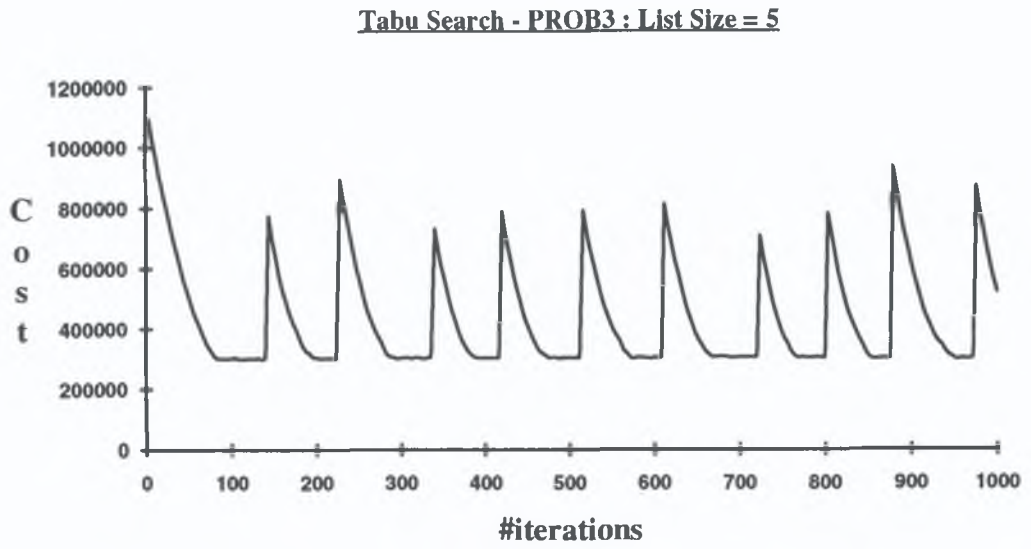


Figure 6.4 *Progression of "Tabu Search 1" for problem "PROB 3" - list size = 5*

Observations:

(i) Note that by choosing a new (random) starting solution on cycling, we effectively have a series of separate runs with different starting solutions.

(ii) Cycling occurs less often when the list size is larger (i.e. 20) but having a smaller list produces somewhat better results. This is because a more broad search is performed with a number of runs; the benefit of this outweighs that of longer individual searches.

(iii) Note that less time is required to achieve good solutions with the test problems where fixed link costs are relatively low (PROB 1, PROB 4) than where they are relatively high (PROB 3, PROB 6).

6.4 COOLING SCHEDULES FOR SIMULATED ANNEALING

The implementation of a finite-time cooling schedule for simulated annealing was described in Section 5.7.1. The algorithm depends on three parameters, α, n, and n_{max}. Recall that α represents the factor applied to the cooling parameter every n transitions or n_{max} iterations, whichever is the smaller. Best results were consistently achieved in our experiments, and in the literature, for values of α in the range $[0.9, 0.99]$. n is chosen to control the speed of convergence and n_{max} is taken as twice n.

Tables 6.4 (i)-(vi) display results for each of the following four parameter settings. The speed was expected to increase by a factor of approximately 10 from one implementation to the next over those presented. As before, each run was performed with a random initial solution as well as a solution with all possible links in place.

Algorithm	α	n	n_{max}	Comment
Sim. Ann. 1	0.99	1 000	2 000	slow
Sim. Ann. 2	0.99	100	200	medium
Sim. Ann. 3	0.99	10	20	fast
Sim. Ann. 4	0.9	10	20	very fast

Thus with "Sim. Ann. 1", for example, the cooling parameter is multiplied by 0.99 after 1000 transitions or 2000 iterations have occurred.

Stopping criterion:

Stop when one of the following occurs:

- 200 000 iterations produce no change in cost
- cooling parameter, c_k, becomes very small (<0.01)

PROB 1	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	8060.0	166 345	166 345
Sim Ann 2	878.2	166 345	166 345
Sim Ann 3	177.5	166 345	166 345
Sim Ann 4	8.5	166 367	166 453

PROB 2	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	6516.0	193 196	193 196
Sim Ann 2	958.2	193 196	193 196
Sim Ann 3	169.8	193 196	193 196
Sim Ann 4	9.3	193 971	193 439

PROB 3	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	5993.2	295 600	295 600
Sim Ann 2	917.1	296 409	299 971
Sim Ann 3	148.4	299 597	305 632
Sim Ann 4	8.8	307 139	311 634

PROB 4	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	13632.0	291 346	291 346
Sim Ann 2	1706.6	291 346	291 346
Sim Ann 3	93.3	291 346	291 346
Sim Ann 4	16.2	291 346	291 346

PROB 5	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	15030 2	393 992	393 992
Sim Ann 2	1822 9	393 992	393 992
Sim Ann 3	89 1	394 059	393 992
Sim Ann 4	17 8	395 128	394 784

PROB 6	Average time	Cost	
Algorithm		Full Initial Solution	Random Initial Solution
Sim Ann 1	12162 0	921 331	921 331
Sim Ann 2	1483 1	922 292	921 331
Sim Ann 3	85 0	943 537	948 147
Sim Ann 4	14 9	938 139	1 033 954

Table 6.4 *Results for various cooling schedules*

Figure 6 5 below shows typical progression of the simulated annealing algorithm (for a fast implementation)

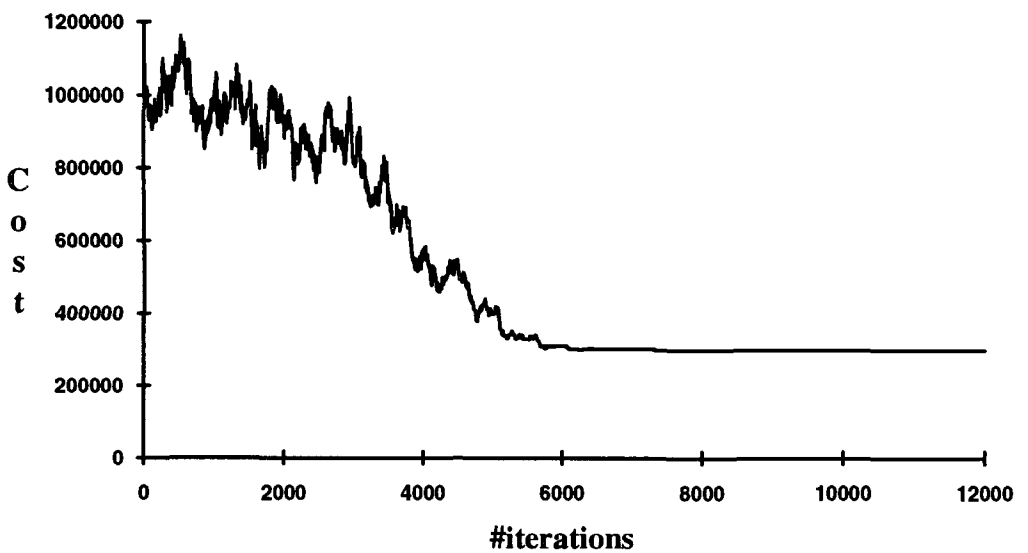


Figure 6.5 *Progression of "Sim Ann 3" for problem "PROB 3"*

Observations:

(i) Cost It can be seen that very good quality solutions are obtainable with the Simulated Annealing algorithm

(ii) Robustness· As expected, the startmg solution chosen has no perceived influence on the quality of solution obtained if sufficient time is allowed for proper progression of the algorithm. Note that the very fast version (Sim Ann 4) is a very crude approximation of simulated annealing

(iii) Time There is a clear correlation between length of time available and quality of solution obtained

(iv) Note that less time is required to achieve good solutions with the test problems where fixed link costs are relatively low (PROB 1, PROB 4) than where they are relatively high (PROB 3, PROB 6)

6.5 GENETIC ALGORITHMS AND AN ADAPTATION

Here we look at the performance of a pure Genetic Algorithm and see what improvements are achieved with our adaptation

6.5.1 Pure Genetic Algorithm

With the simple genetic algorithm, the major tuneable parameters have been identified as

- population size,
- probability of crossover, and,
- probability of mutation

As mentioned in chapter 5, another research project at this university has focused on the application of genetic algorithms to our type of problem This work is reported in [31] A section on parameter tuning identifies the following settings as appropriate for our 20-node problems:

Population of solutions	100
Prob of crossover	1 0
Prob. of mutation	0 0015 per link per iteration

These settings are used for the pure genetic algorithm in the overall comparison of Table 6 7

6.5.2 Hybrid Greedy-Genetic Algorithm

In this section we are concerned with evaluating the adaptation presented in Section 5 8 2 For reasons of clarity, we initially concentrate on one of our test problems, PROB 3, for parameter tuning

A variety of 50-iteration tests were performed on PROB3, with differing expected numbers of crossovers and mutations per iteration. Results are presented in Tables 6.5 and 6.6 below.

Cost:		**Average number of Mutations per iteration**					
		1	**3**	**5**	**7**	**10**	***Average***
Avg.	**1**	297056	296310	295279	295279	295279	*295841*
no. of	**3**	295910	295279	295279	297655	295279	*295880*
cross-	**5**	296209	295279	297017	295279	295279	*295813*
overs/	**7**	297743	295279	295279	295910	295279	*295898*
iter.	**10**	298172	295279	295279	295279	295279	*295858*
	Average	*297018*	*295485*	*295627*	*295880*	*295279*	*295858*

Table 6.5 *Cost comparison over various parameter settings for the greedy-genetic algorithm*

Time:		**Average number of Mutations per iteration**					
		1	**3**	**5**	**7**	**10**	***Average***
Avg.	**1**	131	212	315	329	496	*297*
no. of	**3**	116	209	291	342	477	*287*
cross-	**5**	114	210	290	348	502	*293*
overs/	**7**	124	206	258	338	494	*284*
iter.	**10**	130	190	264	374	427	*277*
	Average	*123*	*205*	*284*	*346*	*479*	*288*

Table 6.6 *Time comparison over various parameter settings for the greedy-genetic algorithm*

Observations:

(i) Cost: Best cost values are seen when the expected number of mutations is about 10 per iteration and the expected number of crossovers is about 5 per iteration. Note though that cost variations are quite small over the range of settings tested.

(ii) Time: The time taken increases with the average number of mutations and stays roughly constant regardless of the crossover rate.

(iii) Robustness: As all tests use random starting solutions, and none produce poor results, this method can be considered robust.

Table 6.7 shows a comparison between results of two variations of this algorithm with a pure genetic algorithm over our six test problems. Random initial solutions are used for all tests.

The following parameter settings were used

- Pure Gen Alg. 5000 iterations, population = 100, prob crossover = 1 0, prob mutation = 0 0015
- Greedy-Genetic 1. Our adaptation (avg 5 crossovers/iteration, 10 mutations/iter)
- Greedy-Genetic 2· Our adaptation (avg 5 crossovers/iteration, 3 mutations/iter)

PROB 1: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7979	166 369
Greedy-Genetic Alg. 1	577	166 345
Greedy-Genetic Alg. 2	275	166 345

PROB 2: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7999	193 196
Greedy-Genetic Alg. 1	667	193 196
Greedy-Genetic Alg. 2	256	193 196

PROB 3: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7949	303 267
Greedy-Genetic Alg. 1	502	295 279
Greedy-Genetic Alg. 2	210	295 279

PROB 4: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7492	291 346
Greedy-Genetic Alg. 1	658	291 346
Greedy-Genetic Alg. 2	304	291 346

PROB 5: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7490	393 992
Greedy-Genetic Alg. 1	477	393 992
Greedy-Genetic Alg. 2	279	393 992

PROB 6: Algorithm	Time (s)	Cost
Pure Genetic Algorithm	7471	928 424
Greedy-Genetic Alg. 1	141	924 939
Greedy-Genetic Alg. 2	120	922 938

Table 6.4 *Results for genetic algorithm variations*

6.6 OVERALL COMPARISON OF LOCAL SEARCH METHODS

Results are presented in Table 6 7 for two implementations of each of the four major algorithms that we are concerned with The result of a random search in the solution space is also shown for comparison purposes This search selects the best of 100 000 randomly chosen solutions

The following is a summary of the parameter tunings chosen for this comparison

Greedy Algorithms.

Acc Greedy : Accelerated greedy algorithm with link removal only

Gr with ins · 'Normal' greedy algorithm with link insertion and removal

Tabu Search

Tabu Search 1 · list size = 5 1000 iterations

Tabu Search 2 · list size = 5 10000 iterations

Simulated Annealing.

Sim Ann 1 . $\alpha = 0\ 99$, $n = 1000$, $n_{max} = 2000$

Sim Ann. 2 . $\alpha = 0\ 99$, $n = 100$, $n_{max} = 200$

Genetic Algorithm

Pure Gen Alg 5000 iterations, pop = 100, prob crossover = 1 0,
prob mutation = 0 0015

Hybrid Greedy-Genetic 2 Our adaptation (avg 5 crossovers/iteration, 3 mutations/iter)

PROB1 : 20 nodes $k_{char} = 0\ 1$ $\xi = 0\ 8$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	0 6	166 345	217 526
Greedy with ins.	31 2	166 345	166 345
Sim Ann 1	8060 0	166 345	166 345
Sim Ann 2	878 2	166 345	166 345
Tabu Search 1	~500 0	166 345	166 345
Tabu Search 2	~5500 0	166 345	166 345
Pure Gen. Alg.	7979 4	N/A	166 369
Greedy-Genetic 2	275 0	N/A	166 345
Rand. Search	~2000 0	N/A	177 156

PROB2 : 20 nodes $k_{char} = 1\,0$ $\xi = 0\,8$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	0 9	193 196	243 762
Greedy with ins.	40 5	193 196	193 955
Sim Ann 1	6516 0	193 196	193 196
Sim Ann 2	958 2	193 196	193 196
Tabu Search 1	~500 0	193 196	193 196
Tabu Search 2	~5500 0	193 196	193 196
Pure Gen. Alg.	7998 8	N/A	193 196
Greedy-Genetic 2	256 0	N/A	193 196
Rand. Search	~2000 0	N/A	222 959

PROB3 : 20 nodes $k_{char} = 10\,0$ $\xi = 0.8$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	1 6	298 687	373 887
Greedy with ins.	42 9	299 328	305 362
Sim Ann 1	5993 2	295 600	295 600
Sim Ann 2	917 1	296 409	299 971
Tabu Search 1	~500 0	297 542	296 613
Tabu Search 2	~5500 0	295 600	295 600
Pure Gen. Alg.	7949 0	N/A	303 267
Greedy-Genetic 2	210 0	N/A	295 279
Rand. Search	~2000 0	N/A	624 723

PROB4 : 20 nodes $k_{char} = 0\,1$ $\xi = 0\,3$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	0 6	291 346	420 126
Greedy with ins.	13 0	291 346	291 346
Sim Ann 1	1706 6	291 346	291 346
Sim Ann 2	93 3	291 346	291 346
Tabu Search 1	~500 0	291 346	291 346
Tabu Search 2	~5500 0	291 346	291 346
Pure Gen. Alg.	7492 2	N/A	291 346
Greedy-Genetic 2	304 0	N/A	291 346

PROB5 : 20 nodes $k_{char} = 1\,0$ $\xi = 0\,3$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	0 8	394 059	529 888
Greedy with ins.	16 8	394 059	394 560
Sim Ann 1	15030 2	393 992	393 992
Sim Ann 2	1822 9	393 992	393 992
Tabu Search 1	~500 0	393 992	393 992
Tabu Search 2	~5500 0	393 992	393 992
Pure Gen. Alg.	7489 5	N/A	393 992
Greedy-Genetic 2	279 0	N/A	393 992

PROB6 : 20 nodes $k_{char} = 10\,0$ $\xi = 0.3$

Algorithm	Average time (s)	Cost	
		Full Initial Solution	Random Initial Solution
Acc. Greedy	1 2	922 938	1 293 905
Greedy with ins.	15 8	935 775	1 088 054
Sim Ann 1	12162 0	921 331	921 331
Sim Ann 2	1483 1	922 292	921 331
Tabu Search 1	~500 0	921 331	921 331
Tabu Search 2	~5500 0	921 331	921 331
Pure Gen. Alg.	7471 4	N/A	928 424
Greedy-Genetic 2	120 0	N/A	922 938

Table 6.7 *Overall comparison of local search methods*

A discussion of these overall results is given with overall conclusions in the next chapter

Chapter 7. Conclusion

This chapter is concerned with summarising the observations and conclusions that arise from this research work. Firstly, Section 7.1 is concerned with a discussion of experimental results. Section 7.2 discusses possible directions for applications and for further research. Finally, Section 7.3 sums up the principal conclusions.

7.1 Overall Discussion of Results

7.1.1 Comparison of Local Search Methods

A summary of the major results has been presented in tabular form in Section 6.6. As previously mentioned, our primary concerns in evaluating the algorithms are cost, robustness and speed.

The best solutions in terms of cost are consistently the modern local search methods. No single one of the three such methods (simulated annealing, tabu search, adapted genetic algorithm) stands out from the others in terms of achievable minimum cost. Although the lowest cost solutions are provided by these modern methods, it is fair to say that all methods looked at (with the exception of course of purely random search) are capable of converging to reasonably good solutions if the conditions are right. Under these conditions (judicious choice of starting solution, for example), each method produces solutions costing no more than one or two percent above the lowest found. In large scale network design, of course, this one or two percent could represent a significant amount of money, and considerable effort to achieve the *best* method or methods would be warranted.

The modern algorithms are more robust than traditional greedy algorithms in two respects. Firstly, the greedy algorithms are highly dependent on the initial solution chosen. This problem is surmountable in the case of the fixed charge problem as choice of a particular solution (full base graph) always gives good results; however, this cannot be guaranteed to work when the cost function is somewhat less well-behaved. Secondly, the performance of greedy algorithms tends to deteriorate as the parameter

k_{char} is increased, causing more diverse local optima to be present (thus trapping the algorithm).

Our third evaluation criterion is speed. Here, the accelerated greedy algorithm is outstandingly fast. Simulated annealing, on the other hand, requires a lot of time for convergence if we wish to be very confident of finding an optimal solution. The solution quality obtainable is directly related to the length of time available. This is also true for tabu search, where considerable time is required in our tests. Our combination of greedy and genetic algorithms is slower than a greedy algorithm but much faster than the other modern methods, while managing to match them in terms of solution quality (cost) and robustness.

7.1.2 Effects of k_{char} on performance

It is clear from the results and the above discussion that the effectiveness of greedy algorithms for the fixed charge problem diminishes with increasing relative significance of initial (fixed) link costs. We can conclude that there exist an increasing number of diverse local optima as k_{char} is increased. The quality of local optimum found then depends on the initial solution chosen.

7.1.3 Conclusions from experimental work

Each of the algorithms presented could be of use for large scale network planning. Choice of algorithm would depend on user-imposed conditions.

The accelerated greedy algorithm would be the best choice for a decision support system, where rapid convergence is just as important as solution quality. It is frequently the case that a designer would like to have an interactive system with the possibility to optimise, make desired changes, re-optimise, and so on, without having to wait around for too long.

Simulated annealing or tabu search would be a better choice in long-term large scale network planning. Here, computation time of the order of hours would be available for finding good solutions. The savings achievable could be quite significant for high cost systems.

The hybrid greedy/genetic algorithm achieves both speed and solution quality and could conceivably be applied in either of the above mentioned scenarios, although it might be a

little slow for interactive decision support It is the closest we have, however, to a general purpose algorithm for this kind of problem

7.2 APPLICATIONS AND SCOPE FOR FURTHER WORK

In this section, we discuss applications arising from this work and the potential for further development.

7.2.1 Direct application to network planning

Two approaches to network planning were mentioned in the previous section interactive decision support and non-interactive long-term planning In this work we have investigated a range of algorithms which could be used in either of these approaches In fact our accelerated greedy algorithm implementation has been used successfully in the GSAC (*G*eneration and *S*election of *A*lternative *C*onfigurations) tool, developed for the collaborative European Union RACE[1] II project, DESSERT[2]

7.2.2 Application of algorithms in other areas

As well as the direct application of these methods for planning of public or private networks, it is worth noting that a wide range of diverse optimisation problems in telecommunications have much in common and can be modelled combinatorially Thus the successful application of these modern methods for the problems we have looked at should encourage their use elsewhere Just one example of another large-scale combinatorial problem in the area of telecommunications is the sizing and location of cells in cellular networks

7.2.3 User interface development

It is generally useful in computer-aided design to have some user interaction with the design process as, for many problems, humans have a superior ability to visualise what a good solution should look like It would be desirable if our design process were to produce a graphical output that the user could change by adding or removing links, being informed of the marginal cost of his actions in the process

---

[1]*R*esearch in *A*dvanced *C*ommunications in *E*urope

[2]*De*cision *S*upport Systems for *Ser*vice Management. This project was completed in December 1994

A very suitable basis for a user interface is a Geographic Information System (GIS) with points on a map representing nodes Currently, the inputs (demands and cost function parameters) are read from a data file in our implementation As cost function parameters are expected to relate closely to distance, the possibility is there for their automatic generation The possibility is also there for incorporating the automatic receipt of demands from the forecasting process The output of our process (link arrangement and capacities) could then be passed back to the GIS for display A prototype implementation of this interaction between a network planning module and a GIS has been achieved by a colleague at this University

7.2.4 Further refinement of the algorithms

As all the local search methods considered need to be tuned, there is limitless scope for their further refinement Algorithms can be adapted to suit the characteristics of various problems by numerical analysis There is considerable scope also for developing new hybrid algorithms like our greedy algorithm/genetic algorithm combination

7.2.5 Other methods

Not all methods for the solution of combinatorial problems have been investigated in this work An interesting approach that is receiving growing attention is Constraint Logic Programming (CLP) Traditional logic programmmg allows for the declarative formulation of combinatorial problems in such a way that the search space is enumerated automatically. Constraints are then used *passively* to test the generated values This is prohibitively slow, however, for large problems, CLP, by contrast, allows the *active* treatment of constraints Constraints are *propagated* to reduce the search space and hence the combinatorial explosion There are a number of academic and commercial CLP languages, the foremost being CHIP (*C*onstraint *H*andling *I*n *P*rolog).

7.3 SUMMARY OF OVERALL CONCLUSIONS

All the general local search methods under consideration have been successfully implemented to a computationally complex network planning problem analysed by Minoux in his paper [30] Considerable efficiencies have been achieved for our implementation by our development of a new approach for updating all the shortest paths from iteration to iteration

We can conclude from our experimental results that modern local search methods, well implemented, produce solutions of better quality than traditional methods

The drawback of the modern methods is their speed This may or may not be a major problem, depending on the length of time available to the user In any case, this problem can be greatly lessened by implementing our hybrid greedy-genetic algorithm.

The performance of traditional methods is poorest for problems with a large number of diverse local optima The number of local optima increases for the fixed charge problem as k_{char}, the ratio of fixed to initial link costs in the optimal solution, increases It has been observed [20] that k_{char} will usually be relatively high in real network planning problems

References

[1] E H L Aarts and J Korst, *Simulated Annealing and Boltzmann Machines*, Wiley, 1989

[2] J W. Billheimer and P Gray, "Network Design with Fixed and Variable Cost Elements", *Transportation Science*, vol 7, pp 49-74, 1973

[3] R R Boorstyn and H Frank, "Large-Scale Network Topological Optimization", *IEEE Trans Commun , vol COM-25*, pp 29-47, January 1977

[4] D D Botvich, "Optimisation of Static Models for Cross Connect Networks", *Unpublished manuscript*, Dublin City University, 1992

[5] W-K Chen, *Theory of Nets Flows in Networks*, Chapter 2, Wiley, 1990

[6] U Dorndorf, "A Short Note on Local Search for DESSERT", *Unpublished manuscript,* DESSERT Project, 1992

[7] M R Garey and D S Johnson, *Computers and Intractability A Guide to the Theory of NP-Completeness*, Freeman, 1979

[8] B Gavish et al, "Fiberoptic Circuit Network Design Under Reliability Constraints", *IEEE J Select Areas Commun* , vol SAC-7, pp 1181-1187, October 1989

[9] M Gerla and L Kleinrock, "On the Topological Design of Distributed Computer Networks", *IEEE Trans Commun , vol COM-25*, pp 48-60, January 1977

[10] A. Gersht and R Weihmayer, "Joint Optimization of Data Network Design and Facility Selection", *IEEE J Select Areas Commun* , vol SAC-8, pp 1667-1681, December 1990

[11] F Glover "Tabu Search - Part I", *ORSA Journal on Computing,* vol 1, pp 190-206, 1989

[12] F Glover "Tabu Search - Part II", *ORSA Journal on Computing,* vol 2, pp 4-32, 1990

[13] F Glover and H J Greenberg, "New approaches for heuristic search· A bilateral linkage with artificial intelligence", *European J Operational Research,* vol 39, pp 119-130, 1989

[14] F Glover, D D Klingman, N V Phillips, R F Schneider, "New polynomial shortest path algorithms and their computational attributes", *Management Science*, vol 31, pp 1106-1128, September 1985

[15] F Glover and M Laguna, "Tabu Search", *Modern Heuristic Techniques for Combinatorial Problems*, C Reeves, ed , Blackwell Scientific Publishing, pp 70-141, 1993

[16] G M Guisewite and P M Pardalos, "Minimum Concave-Cost Network Flow Problems· Applications, Complexity, and Algorithms", *Annals of Operations Research*, vol 25, pp 75-100, 1990

[17] A Hertz and D de Werra, "The tabu search metaheuristic How we used it", *Annals of Mathematics and Artificial Intelligence*, vol 1, pp 111-121, 1990

[18] D S Johnson, J.K Lenstra and A H G Rinnoy Kan, "The Complexity of the Network Design Problem", *Networks*, vol 8, pp 279-285, 1978

[19] D S Johnson, C H Papadimitriou and M Yannakakis, "How easy is local search", *J Computer and System Sciences*, vol 37, pp 79-100, 1988

[20] M Kerner, H L. Lemberg and D M Simmons, "An Analysis of Alternative Architectures for the Interoffice Network", *IEEE J Select Areas Commun* , vol SAC-4, pp 1404-1413, December 1986

[21] A Kershenbaum, P Kermani and G Grover, "MENTOR An Algorithm for Mesh Network Topological Optimization and Routing", *IEEE Trans Commun* , vol COM-39, pp 503-513, April 1991

[22] S Kirkpatrick, C D Gelatt, Jr and M P Vecchi, "Optimization by Simulated Annealing", *Science*, vol 220, pp 671-680, 1983

[23] B W Lamar, Y Sheffi and W B Powell, "A Capacity Improvement Lower Bound for Fixed Charge Network Design Problems", *Operations Research*, vol 38, pp 704-710, July-August 1990

[24] E L Lawler, *Combinatorial Optimization Networks and Matroids,* Holt, Rinehart & Winston, 1976

[25] D N Lee et al, "Solving Large Telecommunication Network Loading Problems", *AT&T Technical Journal,* pp 48-56, May/June 1989

[26] R Lynch, J Murphy and J Sweeney, "Trunk Network Development in the 90s", *Telecom Éireann Technical Journal,* Issue No 9, Summer 1991

[27] J McGibney, D D Botvich, T Curran, "Modern Global Optimisation Heuristics in the Long Term Planning of Networks", *Proceedings of ATNAC'94*, Melbourne, Australia, December 1994

[28] M. Minoux, "Multiflots de coût minimal avec fonctions de coût concaves", *Annales des Télécommunications*, vol 31, pp 77-92, 1976

[29] M Minoux, "Accelerated greedy algorithms for maximising submodular set functions", *Proc IFIP* (J Stoer, Ed), pp 234-243, Springer-Verlang, 1977

[30] M Minoux, "Network Synthesis and Optimum Network Design Problems Models, Solution Methods and Applications", *Networks*, vol 19, pp 313-360, 1989

[31] D O'Meara, *Cost Minimisation of a Telecommunications Network using Genetic Algorithms*, M Eng Project Report, Dublin City University, 1993

[32] C H Papadimitriou and K Steiglitz, *Combinatorial Optimization Algorithms and Complexity*, Prentice-Hall, 1982

[33] A D Pearman, "The Structure of the Solution Set to Network Optimisation Problems", *Transportation Research,* vol 13B, pp 81-90, 1979

[34] G J E Rawlins, *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, 1991

[35] B Yaged, "Minimum Cost Routing for Static Network Models", *Networks*, vol 1, pp 139-172, 1971

[36] M Yannakakis, "The analysis of local search problems and their heuristics", *Proc 7th Annual Symposium on Theoretical Aspects of Computer Science* (C Choffrut and T Lengauer, eds), *Lecture Notes in Computer Science,* no 415, pp 298-311, 1990

[37] N Zadeh, "On Building Minimum Cost Communication Networks", *Networks*, vol 3, pp 315-331, 1973

Appendix 1 Shortest Path Algorithms

A1.1 SHORTEST PATH BETWEEN TWO SPECIFIC NODES: DIJKSTRA'S ALGORITHM

Notation We wish to determine the shortest path between nodes s and t Let $a(i,j)$ denote the distance from node i to node j

Formal statement

ALGORITHM A1.1: DIJKSTRA'S SINGLE-PAIR SHORTEST PATH ALGORITHM

(a) All nodes are initially unlabelled Assign a number $d(x)$ to each node to denote the tentative length of the shortest path from s to x that uses only labelled nodes as intermediate nodes
Initially, let $d(s)=0$ and $d(x)=\infty$, $\forall$ nodes $x \neq s$
Let y denote the last node that was labelled Label node s and let $y=s$

(b) For each unlabelled node x, recalculate $d(x)$ as

$$d(x)=\min\{d(x), d(y)+a(y,x)\} \qquad \text{(A1 1)}$$

If $d(x)=\infty$ for all unlabelled nodes, STOP, no path exists between s and t
Otherwise label node x corresponding to the smallest value of $d(x)$ Also label the link to node x from the labelled node that determined the smallest value of $d(x)$ above
Let $y=x$

(c) If node t has been labelled, STOP The shortest path between s and t is the unique set of labelled links from s to t The length of this path is $d(t)$
Otherwise, repeat step (b)

A1.2 SHORTEST PATHS BETWEEN ALL NODE PAIRS: FLOYD-WARSHALL ALGORITHM

Notation Number the nodes 1, ,N Let d_{ij}^{k} denote the length of the shortest path from node i to node j, where only the first k nodes are allowed to be intermediate nodes If no such path exists, let $d_{ij}^{k} = \infty$ We thus have the following

- d_{ij}^{0} denotes the length of the shortest direct link from i to j, if one exists,
- $d_{ii}^{0} = 0$, for all nodes $i \in 1, \ ,N$, and
- d_{ij}^{N} represents the length of the shortest path from i to j

Let D^{k} denote the $N \times N$ matrix with elements d_{ij}^{k} Our link configuration and distance matrix, $A = (a(i,j))_{i,j \in 1, \ ,N}$, gives D^{0} and our objective is to determine D^{N}

Formal statement

ALGORITHM A1.2: FLOYD-WARSHALL ALL SHORTEST PATHS ALGORITHM

(a) Determine the matrix D^{0} Let $d_{ij}^{0} = a(i,j)$, if link (i,j) is present,
$= \infty$, otherwise

Let $d_{ii} = 0, \forall i$

(b) For $k = 1, \ ,N$, successively determine the elements of D^{k} from the elements of D^{k-1} using the following recursive formula

$$d_{ij}^{k} = \min\left\{d_{ik}^{k-1} + d_{kj}^{k-1}, d_{ij}^{k-1}\right\} \qquad \text{(A1 2)}$$

As each element is determined, record the path that it represents

(c) STOP The matrix D^{N} represents the solution
